Berufsakademie Sachsen
Staatliche Studienakademie Dresden
Studiengang Informationstechnik

Absicherung einer professionellen Vault Software mittels Confidential Computing

Bachelorarbeit
zur Erlangung des Abschlusses Bachelor of Engeneering
im Studiengang Informationstechnik

Wolf, Marvin Thomas
T-Systems Multimedia Solutions GmbH
Matrikelnummer: 3003714

Gutachter: Dr.-Ing. Ivan Gudymenko
Gutachter: Prof. Dr. habil. Andriy Luntovskyy

Tag der Themenübergabe: 30. April 2022
Tag der Einreichung: 15. Juli 2022

Abstract

Wolf, Marvin Thomas: Absicherung einer professionellen Vault Software mittels Confidential Computing, T-Systems Multimedia Solutions GmbH, Berufsakademie Sachsen, Staatliche Studienakademie Dresden, Studiengang Informationstechnik, Bachelorthesis 2022.

79 Seiten, 98 Literaturquellen, 19 Anlagen.

Die Nutzung von Cloud-Infrastrukturen zur Verarbeitung hat sich in den letzten zehn Jahren stark ausgeweitet und wird vielseitig eingesetzt. Die Verlässlichkeit und Kosteneffizienz der Ressourcen sind beispielsweise Vorteile dieser Technik. Es existieren inakzeptable Risiken, hinsichtlich der Integrität und Vertraulichkeit, bei der Verarbeitung von vertrauenswürdigen Daten, wenn Systeme verwendet werden, welche durch eine dritte Partei bereitgestellt werden.

Durch Intel SGX können Anwendungen abgesichert ausgeführt werden, dies wird als Confidential Computing bezeichnet. Der Betreiber des Systems hat keine Möglichkeit auf die verarbeiteten Daten zuzugreifen. Zum einen kann eine Anwendung speziell für die Nutzung von Intel SGX entwickelt werden, zum anderen kann eine bestehende Anwendung migriert werden, um die Funktionalitäten dieser Technologie zu nutzen.

Der Zugang zu Systemen in einem Unternehmen basiert meist auf kryptografischen Schlüsseln oder anderen Geheimnissen, diese dienen der Authentifizierung. Bestimmte Prozesse können das Zurückrufen oder das erneute Ausrollen von diesen Geheimnissen erfordern. Der Konfigurations- und Verwaltungsaufwand steigt mit zunehmender Mitarbeiteranzahl. Die Nutzung einer modernen Schlüsselverwaltungslösung, wie HashiCorp Vault, automatisiert viele Vorgänge und verbessert die Administration deutlich.

Im Rahmen dieser Arbeit soll Intel SGX als eine Confidential Computing Technologie genauer betrachtet werden und HashiCorp Vault durch die Nutzung dieser Technik abgesichert ausgeführt werden. Zusätzlich werden Komponenten eingesetzt, um die Ressourcennutzung zu überwachen und einen externen Zugriff zu ermöglichen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AEs Architectural Enclaves

AES Advanced Encryption Standard

AESM Architectural Enclave Service Manager

AEX Asynchronous Enclave Exit

API Application Programming Interface

APT Advanced Packaging Tool

BIOS basic input/output system

CA Certificate Authority

CCC Confidential Computing Consortium

CD Continous Delivery

CI Continous Integration

CNCF Cloud Native Computing Foundation

CPU Central Processing Unit

DCAP Data Center Attestation Primitives

DES Data Encryption Standard

EPC Enclave Page Cache

EPCM Enclave Page Cache Map

EPID Enhanced Privacy ID

GCM Galois Counter Mode

gRPC gRPC Remote Procedure Calls

HCL HashiCorp Configuration Language

HSM Hardware Security Module

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IAS Intel Attestation Service

iKGF Intel Key Generation Facility

IoT Internet of Things

ISR Interrupt Service Routine

ISRG Internet Security Research Group

MEE Memory Encryption Engine

ML Machine Learning

NIST National Institute of Standards and Technology

OE Owning Enclave

PKI Public Key Infrastruktur

PRM Processor Reserved Memory

PSW Intel SGX Plattformsoftware

QoS Quality of Service

REST Representational State Transfer

RPC Remote Procedure Call

RSA Rives-Shamir-Adleman

SDK Software Development Kit

SEV Secure Encrypted Virtualization

SEV-ES SEV with Encrypted State

SGX Software Guard Extensions

SME Secure Memory Encryption

SNP Secure Nested Paging

SoC System-on-a-Chip

SRC Secure Remote Computation

SSL Secure Sockets Layer

TCB Trusted Computing Base

TDX Trust Domain Extensions

TEE Trusted Execution Environment

TLB Translation Lookaside Buffers

TLS Transport Layer Security

VCS Version Control System

WPA2 Wi-Fi Protected Access 2

Tabellenverzeichnis

1 Einführung

Unternehmen legen eher selten viel Wert auf die Absicherung von Infrastrukturen und die Überprüfung der internen Sicherheit. Meist wird dies nicht absichtlich getan und man sieht potenzielle Risiken nicht. Werden gewisse Dienste intern bereitgestellt und genutzt, so ist es notwendig, dass eine Authentifizierung vorhanden ist. Diese kann mittels Zertifikaten geschehen. Es kommt zu Sicherheitsproblemen innerhalb der Unternehmen, wenn keine zeitgemäßen Verfahren und Werkzeuge eingesetzt werden. Damit Zertifikate und Schlüssel schnell und einfach von den Administratoren verwaltet werden können, ist der Einsatz einer Schlüsselverwaltungslösung notwendig.

Um Sicherheitslücken innerhalb der Software selbst zu entgehen, sollte auf sogenannte Open-Source-Software gesetzt werden. Bei dieser ist der Quellcode der Anwendung öffentlich zugänglich. Entwickler können Änderungen vorschlagen und die Arbeit anderer überprüfen. Wird ein Konsens über die vorgeschlagenen Änderungen erreicht, wird sie in die Anwendung übernommen und somit in die finale Lösung. Das Risiko, dass Sicherheitslücken vorhanden sind, wird dadurch minimiert.

Um die Wahrscheinlichkeit von Sicherheitslücken bei der Nutzung eines kritischen Dienstes, wie eine Schlüsselverwaltungssoftware, zu verringern, ist es ratsam eine Open-Source-Lösung zu verwenden, wie beispielsweise HashiCorp Vault.

Die Verwendung der Open-Source-Lösung Vault ist alleine aber nicht ausreichend, da dem System, auf welchem die Anwendung ausgeführt wird, vertraut werden muss. Wird ein System innerhalb des Rechenzentrums eines Unternehmen zur Ausführung genutzt, besteht das Risiko, dass Angreifer welche erweiterte Rechte erlangt haben, Daten mitlesen können während der Verarbeitung. Wird die Anwendung in einer Cloud-Umgebung ausgeführt, ist man gezwungen dem Betreiber der Cloud zu vertrauen, dass dieser inklusive seinen Administratoren nicht auf die Daten zugreifen und diese missbrauchen.

Um dem Sicherheitsproblem des Vertrauens in das zugrundeliegende System zu begegnen, wird Confidential Computing eingesetzt. Dabei soll verhindert werden, dass Administratoren oder Personen, welche Zugang zur Hardware haben, Zugriff auf die Daten erlangen und diese verfälschen können.

Die Motivation ist das Entgegenwirken der gerade beschriebenen Risiken von Innentätern und Angreifern, mit privilegierten Rechten. Dabei ist es erforderlich, dass die ununterbrochene Absicherung der Daten und deren Integrität während der gesamten Lebensdauer gewährleistet ist. Daten befinden sich stets in einem der drei folgenden Zustände: (1) während der Übertragung (engl.: *Data in transit*), (2) während der Verarbeitung (engl.: *Data in use*) und (3) gespeichert auf einem Datenträger (engl.: *Data at rest*). Daraus folgend

müssen die Daten in jedem Zustand durch technische Maßnahmen gesichert sein. Für die Fälle (1) und (3) existieren eine große Menge von Werkzeugen und kryptografischen Verfahren. (1) Daten können mittels Transport Layer Security (TLS) verschlüsselt übertragen werden. (3) Durch die Verwendung von Advanced Encryption Standard (AES) oder Rives-Shamir-Adleman (RSA) können Daten verschlüsselt auf einem Datenträger abgelegt werden. Der Schutz der Daten während der Verarbeitung stellt eine besondere Herausforderung dar. Dies wird erschwert, wenn diese Verarbeitung auf einem externen System stattfindet, welches für den Nutzer nicht vollständig kontrollierbar ist.

Aufgabenstellung dieser Arbeit ist die Verwendung von Confidential Computing zur Absicherung der Funktionalitäten der Open-Source-Lösung HashiCorp Vault, um die beschriebenen Sicherheitsprobleme zu lösen.

2 Vorstellung HashiCorp Vault Software-Lösung

Die Software-Lösung Vault wird vom amerikanischen Unternehmen HashiCorp entwickelt. Die Software wurde vorrangig in der Programmiersprache Go geschrieben. Der Quellcode ist vollständig Open-Source und auf GitHub verfügbar. Es gibt zwei verschiedene Wege Vault zu nutzen, entweder man installiert es selbst und übernimmt die komplette Konfiguration oder man nutzt eine fertige Vault-Installation, HashiCorp Cloud Platform. Bei diesem Ansatz übernimmt HashiCorp die komplette Installation, Konfiguration und stellt die Verfügbarkeit sicher. Dieser Dienst ist kostenpflichtig.[1]

Herkömmliche On-Premise-Infrastrukturen sind meist sehr statisch und ihre Sicherheit beruht auf separaten Servern und einem klar abgesteckten Netzumfang mit statischen IP-Adressen, sodass Server nur Anfragen aus einem bestimmten IP-Adressbereich entgegen nehmen können. Durch die Umstellung zu einer dynamischen Infrastruktur, unter Umständen mit verschiedenen Anbietern, muss der Sicherheitsansatz neu betrachtet werden.[2] Die Sicherheit entsteht durch meist kurzlebige Anwendungen, sowie eindeutigen vertrauten Quellen für die Identitäten der Nutzer und Anwendungen. Zudem findet meist eine software-basierte Verschlüsselung statt. Durch die Nutzung der Cloud werden bei Bedarf neue Instanzen bereitgestellt, um die zusätzliche Last abzufertigen. In aller Regel werden in diesen dynamischen Strukturen Benutzeraccounts geteilt, um Zugriff auf APIs oder Datenbanken zu erhalten. Ebenso werden Schlüssel zur Verschlüsselung oder Kommunikation mit Dateisystemen geteilt. Um die Instanz beim Aufsetzen zu konfigurieren, werden zentrale Konfigurationsdateien genutzt, in diesen ist dann das Passwort oder der Schlüssel der Benutzeraccounts gespeichert. Diese Konfigurationsdateien können durch Dritte ausgelesen werden.

Häufig wird einer der folgenden zwei Wege genutzt, um Administratoren die entsprechenden Rechte zu gewähren. Entweder wird der Zugang zu einem einzigen Administratorkonto vom Team geteilt oder alle Mitglieder des Teams erhalten die Administratoren-Rechte. Beide Fälle haben Nachteile und bürgen ein Sicherheitsrisiko, entweder kennt ein Mitglied die Zugangsdaten noch, wenn es das Unternehmen verlässt oder die Abteilung wechselt oder die Administratoren mit den ausgeweiteten Rechten nutzen den gleichen Account für ihre Aufgaben im Alltag.

Die beiden, gerade beschriebenen, Szenarien zeigen, dass unterschiedliche Instanzen ein und die selben Schlüssel und Geheimnisse nutzen, um die Arbeit zu erledigen, entweder mehr Last abzufertigen oder die Administration. Daraus ergibt sich das Problem der Zurechenbarkeit der durchgeführten Arbeiten. Es kann schlecht nachvollzogen werden, wer sich auf einem entsprechenden System angemeldet hat. Greifen verschiedene Systeme

[1] Vgl. HashiCorp, Inc (2022f).
[2] Vgl. Bundesamt für Soziale Sicherung (2021).

in der Cloud, mit den gleichen Zugangsdaten, auf eine gemeinsame Datenbank zu, so kann im Zweifelsfall nicht eindeutig festgestellt werden, welches System eine schad- oder fehlerhafte Aktion durchgeführt hat. Zusätzlich werden in Umgebungen wie Kubernetes oder Service Mesh Token benötigt. Ungünstigerweise werden diese in der Regel ebenso in Konfigurationsdateien abgelegt, was auch wieder ein Sicherheitsproblem darstellt.

Bei der Entwicklung von Anwendungen oder Diensten wird meist eine Continous Integration (CI)-/ Continous Delivery (CD)-Pipeline genutzt, um Software zu verteilen und zu testen, nachdem Veränderungen im Code vorgenommen wurden. Zusätzlich wird ein Version Control System (VCS) genutzt, wie GitHub, um Änderungen am Code nachvollziehen zu können. Meist werden nicht die Zugänge zu externen Diensten oder Dateisystemen verändert, sodass die Zugangsdaten dafür gespeichert sind und wieder verwendet werden.

Im Zuge der Entwicklung muss eine klare Trennung zwischen Test-/ Entwicklungssystemen und Produktivsystemen gegeben sein, sodass beispielsweise keine fehlerhaften Änderungen an Kundendaten vorgenommen werden können. Ein weiterer Sicherheitsaspekt ist, was mit den Geheimnissen passiert, wenn die Systeme und Strukturen sich nach der Verteilung verändert haben.

Im Folgendem werden einige Anforderungen an eine moderne Geheimnisverwaltungslösung aufgelistet:

- **Genereller Geheimnisspeicher**
- **Überwachter Speicher für Mitarbeiter**
- **Dynamische API-Schlüssel**
- **Verschlüsselung von Daten**

Vault als Software-Lösung versucht all diesen Anwendungsgebieten zu begegnen. Es ist ein identitäts-basiertes Geheimnis- und Verschlüsselungssystem. Solch ein Geheimnis kann zum Beispiel ein Token, ein Kennwort, ein Zertifikat oder ein Schlüssel sein. Weitere vertrauenswürdige Daten, wie Umgebungsvariablen oder API-Schlüssel, können verschlüsselt abgelegt werden. Dynamische Strukturen benötigen meist Zugriff auf mehrere Geheimnisse in Kombination oder unterschiedliche für die jeweilige zugrundeliegende Plattform, dieser Umstand ist meist schwer zu verstehen und zu durchblicken. An dieser Stelle unterstützt Vault.

Der grundlegende Ablauf der Nutzung von Vault wird im folgenden beschrieben:

1. **Authentifizierung** Ist der Prozess, in welchem ein Nutzer die benötigten Daten bereitstellt, damit Vault die Identität feststellen kann. Bei erfolgreicher Identifizierung wird ein Token generiert und einer Richtlinie zugeordnet.

2. **Validierung** Vault validiert den Nutzer gegenüber einer dritten Partei, beispielsweise GitHub, LDAP oder einer AppRole.

3. **Autorisierung** Der Nutzer wird mit der Vault Sicherheitsrichtlinie verglichen. Diese ist eine Sammlung von Regeln, welche API-Schnittstellen der Nutzer mit seinem eigenen Token nutzen darf, sodass klar festgelegt werden kann, auf welche Daten er Zugriff hat und auf welche nicht.

4. **Zugriff** Vault erlaubt den Zugriff auf die Geheimnisse, Schlüssel und Verschlüsselungsfunktionalitäten, indem ein Token ausgestellt wird, welcher auf Richtlinien der Identität basiert.

Sowohl reelle Nutzer als auch Maschinen oder Dienste können Vault nutzen und treten dabei mit ihrer Identität auf. Vault kann als vertrauenswürdige Instanz genutzt werden, um Anwendungen gegenüber anderen Anwendungen zu attestieren.

In Cloud-Umgebungen können dynamisch Schlüssel und Token an neue Instanzen verteilt werden, sodass diese ein einmalig für jede Instanz sind. Es muss kein Geheimnis im Vorhinein definiert und verteilt werden. Kubernetes kann eine Vault-Installation nutzen, um Geheimnisse mit dieser zu verwalten.

Weitere Funktionen von Vault sind die automatische Rotation von Datenbank-Passwörtern, die Verschlüsselung und Tokenisierung von Daten und die zentrale Unterstützung bei der Nutzung verschiedener Schlüsselverwaltungsanbieter.

Die Verbindung zu Vault wird mittels TLS abgesichert. Jeder Nutzer muss bei jeder Anfrage seinen Token mitsenden. Ein Nutzer, der dies nicht tut, darf nur Anfragen zum Einloggen senden. Die Daten, welche Vault in Richtung des Backends verlassen werden durch Vault mit AES 256-Bit im Galois Counter Mode (GCM) mit 96-Bit Noncen verschlüsselt, zudem wird, je nach verwendetem Backend, TLS zur Kommunikation mit diesem verwendet. Das Backend wird grundsätzlich als nicht vertrauenswürdig betrachtet.[3]

Wird ein Vault Server standardmäßig gestartet, so ist dieser im versiegeltem (engl.: *sealed*) Zustand. Bevor jegliche Operation ausgeführt werden kann, muss dieser entsiegelt werden, da die Daten verschlüsselt sind. Dies geschieht durch die Bereitstellung des Entsiegelungsschlüssel. Der Schlüssel zum Entschlüsseln wird im Schlüsselring mit den Daten gespeichert, er ist aber durch den *root key* verschlüsselt und dieser ist auch bei den Daten

[3] Vgl. HashiCorp, Inc (2022a).

gesichert. Mittels des Schlüssels zur Entsieglung kann der *root key* entschlüsselt werden. Standardmäßig wird das *Shamir's Secret Sharing*[4] Verfahren verwendet, um den *root key* und damit den Entsiegelungsschlüssel in eine vorgegebene Menge aufzuteilen. Zur Rekonstruktion des eigentlichen wird nur eine gewisse Teilmenge dieser Einzelteile und nicht alle benötigt. Die Anzahl aller Teile sowie die benötigte Anzahl kann frei definiert werden. Ebenso kann auf dieses Verfahren verzichtet werden und ein Hardware Security Module (HSM) zum entsiegeln verwendet werden.[5]

HashiCorp Vault bietet verschiedene Wege zur Kommunikation und Interaktion mit dem Vault Server, zum einen kann die Weboberfläche genutzt werden, ebenso ist eine Verbindung mittels Kommandozeilenwerkzeugen möglich oder die Nutzung der HTTP-API.

Vault als Software-Lösung bietet die Möglichkeit eine Public Key Infrastruktur (PKI) bereitzustellen. Sie kann dynamisch X.509 Zertifikate generieren. Die Erstellung dieser wird stark vereinfacht, da viele Schritte durch Vault übernommen werden. Vault übernimmt in diesem Szenario die Rolle der Certificate Authority (CA), welche Zwischenzertifikate generiert und Zertifikate ausstellen und zurückrufen kann.[6]

Zusätzlich kann Vault als *Encryption-as-a-Service* genutzt werden. Dabei werden Daten während der Übertragung ver- bzw. entschlüsselt. Die Vault Application Programming Interface (API) gibt die verschlüsselten oder entschlüsselten Daten zurück, ohne das Vault sie selbst speichert. Die verschlüsselten Informationen können so im Speicher der Anwendung abgelegt werden und es muss kein detailreiches Konzept zur Verschlüsselung implementiert werden.[7]

Vault nutzt wie andere Produkte HashiCorps die eigens entwickelte Konfigurationssprache HashiCorp Configuration Language (HCL). Ihr Fokus liegt auf einem Ausgleich zwischen der einfachen Verständlichkeit für den Menschen und der einfachen und effizienten Verarbeitung durch die Maschine.[8]

Zusammenfassend bietet HashiCorp Vault einen großen Funktionsumfang für viele verschiedene Möglichkeiten und Einsatzzwecke, um Geheimnisse, Schlüssel oder Zertifikate dynamisch und strukturiert zu verteilen für Nutzer und Maschinen. Die unterschiedlichen Möglichkeiten der Authentifizierung und Autorisierung ermöglichen eine prozessuale und technisch abgesicherte Verwendung. Durch die Verwendung des bewährten Shamir-Secret-Sharing-Verfahren kann die Verantwortung auf mehrere Personen zuverlässig verteilt werden.

---

4 Vgl. Shamir, A. (1979).
5 Vgl. HashiCorp, Inc (2022e).
6 Vgl. HashiCorp, Inc (2022d).
7 Vgl. HashiCorp, Inc (2022b).
8 Vgl. HashiCorp, Inc (2022c).

3 Confidential Computing

Immer mehr Unternehmen, aber auch Privatanwender sehen das große Potential in Cloud-Computing und den damit verbundenen Möglichkeiten und Vorteilen. Bereits viele Unternehmen verwenden diese Technologie entweder teilweise oder komplett für ihre Informationstechnische Infrastruktur. Eine in 2019 durchgeführte Studie zeigt, dass 94 % aller befragten Firmen Gebrauch von Cloud-Technologien machen oder deren Einsatz planen beziehungsweise diskutieren.[9]

Durch die Auslagerung von Anwendungen und Ressourcen kann die Kosteneffizienz gesteigert werden. Nichtsdestotrotz birgt die Nutzung von Ressourcen, welche durch den Cloud-Anbieter bereitgestellt werden einige gewisse Risiken. Dem Nutzer müssen diese bereits vor dem Einsatz der Technologie bekannt sein und er muss Möglichkeiten diskutieren, um diese zu minimieren oder ganz zu eliminieren.

Bisherige Maßnahmen aus dem Sicherheitsbereich können zur Verhinderung einiger Risiken verwendet werden. Zum Beispiel können Firewalls oder Application Level Gateways eingesetzt werden, um den Datenverkehr in sogenannten Software-Defined Networks, virtuelle Netzwerke zu überwachen und zu kontrollieren.[10] Außerdem kann die Zwei-Faktor-Authentifizierung verwendet werden um Nutzer zu überprüfen.

Bestimmte Risiken können nicht durch den Einsatz von Technologien verringert werden. Der Nutzer muss auf eine konforme Handlungsweise des Anbieters der Cloud-Umgebung vertrauen. Der Kunde eines solchen Anbieters muss stets davon ausgehen, dass der Betreiber die Möglichkeit besitzt, vollen Zugang zu allen vorhandenen Daten und Informationen in seiner betriebenen Cloud-Umgebung zu bekommen. Da dieser die Hardware, welche für die virtualisierten Ressourcen genutzt wird, kontrolliert. Um diese Technik zu verwalten, beschäftigt er zudem Administratoren. Es besteht keine Möglichkeit unerlaubte Zugriffe zu kontrollieren oder zu vermeiden, für den Nutzer besteht nur die Möglichkeit auf die Einhaltung der vertraglich zugesicherten Rahmenbedingungen zu vertrauen.

Der Nutzer muss sich aber nicht nur vor Angreifern innerhalb, „Innentätern“, des Systems sorgen, sondern auch vor externen Angreifern, welche Zugang zur Cloud-Infrastruktur erlangt haben und damit ähnlich privilegiert sind. Dies kann beispielsweise durch eine Ausweitung der Rechte geschehen sein und so über die Möglichkeiten des Administrators verfügen, in Linux wäre das der Benutzer *root*. Vorhandende IT-Sicherheitswerkzeuge können in diesem Fall recht wenig zur Verhinderung beitragen, da diese in der Regel von den Administratoren konfiguriert und eingesetzt werden. Ein Angreifer mit solchen

[9] Vgl. Heidkamp, P. et al. (2020), S. 37.
[10] Vgl. Luntovskyy, A. (2021a), 197f.

Rechten kann diese dann deaktivieren, um die Tat zu unterstützen, sodass beispielsweise kein Alarm ausgelöst wird oder um Spuren zu beseitigen.

Bei besonders sensiblen Anwendungsszenarien mit streng vertraulichen Daten, wie zum Beispiel Geschäftsgeheimnissen, Patientenakten oder generell bei der Verarbeitung von personenbezogenen Daten stellt dies ein nicht zu akzeptierendes Risiko dar, welches den Einsatz von Cloud-Technologien bisher im Weg steht.[11]

Im Folgendem werden mögliche Anwendungsgebiete vorgestellt, in welchen der Einsatz von Confidential Computing sinnvoll wäre, um die schützenswerte Daten besser abzusichern.

Die Integrität und Vertraulichkeit von Daten ist in vielen verschiedenen Branchen von großer Bedeutung, vor allem auch während der Verarbeitung dieser. Das Medizinwesen wäre eine solche Branche. Zu Forschungszwecken arbeiten meist mehrere Kliniken und Krankenhäuser zusammen, „um ein Machine Learning (ML)-Modell zu entwickeln“[12] und zu verbessern. Während der Bearbeitung müssen die Patientendaten stets sicher sein, sodass die Vertraulichkeit gewahrt wird und keinerlei Rückschlüsse auf die eigentlichen Patienten gezogen werden können.

Eine weitere Branche ist die Fahrzeugindustrie, in dieser wollen Entwickler und Hersteller von Anwendungen Sensordaten aus den Fahrzeugen gewinnen und weiterverarbeiten, dadurch wird dann die eigene Entwicklung unterstützt und verbessert. Mittels bestimmter Technologien bei der Auswertung wird verhindert, dass Rückschlüsse auf den eigentlichen Fahrer durch den Hersteller gezogen werden können. Confidential Computing sichert diese Anonymität, während die Daten verarbeitet werden.

Die Auswertung von Sensordaten erfolgt nicht nur in der Fahrzeugindustrie, sondern auch in Industrieanlagen oder Fabriken. Diese Auswertung und Weiterverarbeitung steigert die Produktivität und Effizienz. Um die Kosten weiter zu senken, kann auf Cloud-Technologien gesetzt werden. Dadurch hat der Anbieter dieser Cloud das Potenzial Zugang zu Firmengeheimnisse oder speziellen technischen Entwicklungen zu erlangen und diese missbräuchlich zu verwenden. Durch die Verwendung von Confidential Computing-Technologien sind die Daten während der Verarbeitung geschützt und das Risiko, dass der Anbieter Zugriff auf sie erlangt, ist nicht vorhanden.

[11] Vgl. Kochovski, A. (2022).
[12] Schuster, F. (2021).

4 Geheimnisverwaltung

In diesem Abschnitt der Arbeit werden grundlegende Geheimnisse in der Informationssicherheit vorgestellt und wofür diese genutzt werden.

4.1 Kryptografische Schlüssel

Die Verschlüsselung von Informationen und Daten ist ein wichtiger Aspekt der Informationssicherheit. Mittels Verschlüsselung wird erreicht, dass sensitive Informationen nur von berechtigten Personen gelesen werden können, sodass die Vertraulichkeit der Informationen gewahrt wird. Dafür wird immer mindestens ein Schlüssel benötigt.

Ende der 70er Jahre wurde in den USA der Data Encryption Standard (DES) als Standard für symmetrische Verschlüsselung bekannt gegeben. Die Schlüssellänge dieses Systems beträgt 56 Bit und ist damit sehr kurz. Es gibt ungefähr 72 Billiarden (2^{56}) mögliche Schlüssel, die verwendet werden können. Mit ausreichend Rechenleistung ist es möglich durch einen Brute-Force-Angriff die Verschlüsselung zu brechen, bei solch einem Angriff werden alle möglichen Schlüssel durch probiert, bis der passende gefunden wird.

Im Oktober des Jahres 2000 wurde AES ausgewählt durch das National Institute of Standards and Technology (NIST), um den Data Encryption Standard zu ersetzen. Joan Daemen und Vincent Rijmen entwickelten den dazugehörigen Algorithmus. Diese Blockchiffre setzt auf eine 128 Bit große Blockgröße. Je nach Anforderung variiert die Schlüssellänge zwischen 128, 192 oder 256 Bit.[13] Aktuell ist kein Angriff bekannt, welcher praktisch durchführbar ist, da sehr viel Zeit benötigt werden würde. Theoretisch gesehen, ist das Verfahren aber gebrochen. Die Vereinigten Staaten von Amerika haben die Standards AES-192 und AES-256 für Dokumente zugelassen, welche den höchsten Grad an Geheimhaltung haben. Die Zahl im Standard gibt die verwendete Schlüssellänge an. Unter anderem wird AES auch bei Wi-Fi Protected Access 2 (WPA2) genutzt. Um die Verschlüsselung und Entschlüsselung zu beschleunigen, gibt es neben der Software-Umsetzung verschiedene Hardware-Implementierungen, sogenannte HSM. Die Verschlüsselung mittels AES funktioniert in sogenannten Runden, bei 128 Bit Schlüssellänge sind es zehn, bei 192 Bit zwölf und bei 256 Bit Schlüsseln 14 Runden. Jede Runde umfasst dabei verschiedene Verarbeitungen, wie die Substitution, das Umstellen und das Durchmischen des ausgehenden Klartexts.

Bei der symmetrischen Verschlüsselung wird der gleiche Schlüssel für das Ver- und Entschlüsseln genutzt, sodass dieser allen Beteiligten bekannt sein muss. Sollen Informationen

[13] Vgl. Daemen, J./Rijmen, V. (2001).

nur von einer Person oder einem kleinem Kreis entschlüsselt werden, wird asymmetrische Verschlüsselung genutzt. Dabei wird eine Information mit dem öffentlichen Schlüssel der zu empfangenden Person verschlüsselt und nur diese Person kann die Information mit ihrem privaten Schlüssel entschlüsseln und so die Informationen lesen.

Das RSA-Verfahren ist das bekanntestes asymmetrische Verschlüsselungsverfahren, zusätzlich kann es zum Signieren von Daten genutzt werden. Das Verfahren ist nach seinen Entwicklern benannt.[14] Meist ist der öffentliche Schlüssel für die Allgemeinheit verfügbar und der private Schlüssel kann nicht aus diesem berechnet werden. Die digitale Signatur funktioniert, indem eine Hashfunktion auf die Nachricht angewendet wird und der daraus entstandene Hash signiert wird. Anschließend kann der Empfänger die Signatur überprüfen, indem er eine spezielle Hashfunktion auf die Nachricht anwendet. Der mathematische Hintergrund von RSA ist die Zerlegung von großen Zahlen in die dazugehörigen Primfaktoren.

Im Hypertext Transfer Protocol Secure (HTTPS)-Protokoll wird TLS verwendet, um Daten sicher zu übertragen. Transport Layer Security hat unter anderem zwei wesentliche Komponenten, zum Einen der TLS Handshake und zum Anderen der TLS Record. Beim Handshake werden die Schlüssel ausgetauscht und es findet eine Authentifizierung statt, meist wird dafür RSA als Kryptosystem eingesetzt. Im TLS Record wird dann der verhandelte symmetrische Schlüssel genutzt, um die Daten vor der Übertragung zu verschlüsseln, dafür kann AES genutzt werden.[15]

Die Kombination der zwei Verfahren, AES und RSA, steigert die Sicherheit und Effizienz der abgesicherten Datenübertragung in Netzwerken. Der Verbindungsaufbau ist auf der Seite des Servers rechenintensiver und deshalb etwas langsamer.

4.2 Zertifikate

Zertifikate stellen einen Datensatz dar, welcher schützenswert ist und eine wichtige Rolle in der Informationssicherheit spielt. Dieser Datensatz enthält Eigenschaften oder Attribute von Menschen oder anderen Objekten. Anhand kryptografischer Verfahren kann die Integrität und Authentizität der bestätigten Informationen überprüft werden. Das Zertifikat enthält alle notwendigen Daten dafür.

Die Zertifikate und damit die öffentlichen Schlüssel werden durch eine Zertifizierungsstelle, der sogenannten CA, ausgegeben. Bei der Public-Key-Infrastructure (PKI), werden Zertifikate beispielsweise von einer Institution ausgestellt und verteilt. Ein anderer Nutzer des

[14] Vgl. Rivest, R. L./Shamir, A./Adleman, L. M. (1978).
[15] Vgl. Internet Engineering Task Force (IETF) (2018).

Systems kann den öffentlichen Schlüssel zu einer Identität zuordnen und damit die Integrität und Vertraulichkeit der Daten überprüfen. Außerdem kann die Signatur verifiziert werden.[16] Die CA stellt eine vertrauenswürdige dritte Partei dar. Mittels Kryptografie lässt sich beweisen, dass ein erhaltenes Zertifikate tatsächlich von dieser CA stammt, sodass die prüfende Person davon ausgehen kann, dass der Gegenüber der rechtmäßige Eigentümer des Zertifikates ist und sich nicht als eine andere Institution ausgibt.

[16] Vgl. Hunt, R. (2001).

5 Cloud- und Containerumgebung

Im folgendem Abschnitt der Arbeit wird das Prinzip des Cloud Computings vorgestellt, weiterhin soll die Container-Technologie und die Orchestrierung von mehreren Containern erläutert werden.

5.1 Cloud-Computing

Unter Cloud Computing versteht man das Modell, dass Anwendungen, Server oder andere Computerressourcen geteilt genutzt werden. Sie werden meist über das Internet und geräteunabhängig verwendet. Die Bereitstellung erfolgt in aller regel zeitnah und schnell. Die Abrechnung geschieht nach der Nutzung beziehungsweise des Verbrauchs. Der Nutzer einer solchen Ressource benutzt meist eine API oder Website zur Interaktion mit dieser.

Es wurden fünf essentielle Eigenschaften und Merkmale durch das *NIST*[17] für das Cloud Computing definiert:

- **On-demand self-service** Leistungen und Ressourcen können durch den Nutzer bei Bedarf bereitgestellt werden lassen, ohne das eine weitere Person handeln muss.
- **Broad network access** Die Nutzung der Ressourcen ist von unterschiedlichen Geräten aus möglich, vor allem über das Netzwerk mit gewissen Standards.
- **Resource pooling** Die Ressourcen eines Anbieters betrachtet man als Gesamtheit, sodass Nutzer, je nach ihrem Bedarf, nach dem sogenannten Mehrmandantenprinzip bedient werden. Er hat keinen Einfluss, welche Ressourcen die geforderten Leistungen erbringen.
- **Rapid elasticity** Die Leistung kann dynamisch zur Verfügung gestellt werden, sodass eine bedarfsgerechte Skalierung nach oben und unten möglich ist. Für den Nutzer erscheinen die Ressourcen unbegrenzt und können beliebig angepasst werden.
- **Measured service** Die Computerressourcen werden ständig überwacht und optimiert durch messbare Werte, wie zum Beispiel der Speicher oder die Bandbreite. Dadurch entsteht eine gewisse Transparenz für den Anbieter und den Nutzer.

Nach dieser Definition wird unter Cloud Computing mehr als herkömmliche Virtualisierung verstanden. Diese Definition findet weitgehend Akzeptanz.

[17] Vgl. Mell, P./Grance, T. (2011).

Viele führende Informatiker gaben zu Beginn der 1990er Jahre einen Ausblick für Cloud Computing. Die Allgemeinheit ging davon aus, dass Computerressourcen sich in einem Netz verteilen würden und gemeinsam genutzt werden können. Gegen Ende dieses Jahrzehntes wurde die sogenannte Multitenant-Architektur entwickelt. Dabei konnten mehrere Kunden gleichzeitig auf demselben Server oder mit der gleichen Software arbeiten, ohne das diese Einblick in die Daten untereinander hatten.

Zu Beginn des 21. Jahrhunderts wuchs die Anzahl der Kunden von Amazon immer weiter und stärker. Gerade in Zeiten, wo Spitzenlasten erreicht wurden, wie zum Beispiel in der Weihnachtszeit, wurde viel Leistung und Performance benötigt. Sodass Amazon begann selbst an einer Lösung des Problems zu arbeiten und entwickelte so eine serviceorientierte Architektur. Durch den Fokus auf die Skalierbarkeit der Dienste wurde diese Entwicklung später zu einem Produkt, welches extern Kunden angeboten wurde. Der so entstandene Dienst heißt Amazon Web Services.[18]

Im Jahr 2008 wurde die Google Cloud Platform gegründet. Sie umfasst einige Cloud-Dienste, welche selbst durch Google genutzt werden. Nutzer haben die Möglichkeit virtuelle Maschinen, Kubernetes Cluster, Speichermedien und viele weitere Strukturen zu erstellen und zu nutzen.[19]

Ebenfalls im Jahr 2008 kündigte Microsoft Azure als Plattform für das Cloud-Computing an. Offiziell ist sie seit dem 1. Februar 2010 verfügbar. Vorrangig ist die Plattform auf Softwareentwickler ausgerichtet. Microsoft setzt derzeit 4 Rechenzentren innerhalb Deutschlands ein, sodass für Deutschsprachige Kunden Datenschutzverordnungen eingehalten werden können. Microsoft entwickelt ständig neue Dienste, wie beispielsweise Service Meshs oder eine zentrale Verwaltung für Internet of Things (IoT)-Systeme.[20]

Ziel des Cloud Computings war zunächst die Reduktion der Kosten und des Verwaltungsaufwandes, da nur die tatsächlich verbrauchte Nutzung abgerechnet wird. Im Laufe der Zeit gewann der Faktor der Flexibilität an Wert. In der heutigen Zeit stehen Cloud Technologien für Innovation und gesteigerte Produktivität.

Entwickler profitieren von Cloud-Umgebungen ungemein, da sich im Cloud-Umfeld verschiedene Umgebungen realisieren lassen. Produktiv- und Testsysteme können separiert von einander in verschiedenen virtuellen Maschinen ausgeführt werden. Möchte beispielsweise ein Entwickler eine aktualisierte Version einer Software testen, so kann dieser sich schnell und einfach eine neue virtuelle Maschine starten, die Anwendung ausgiebig testen und nach erfolgtem Testen wieder die virtuelle Maschine löschen. Dieser Vorgang ist sehr kosten- und zeiteffizient, da nur die über die Zeit genutzten Ressourcen berechnet werden.

[18] Vgl. Amazon Web Services (2022).
[19] Vgl. Paul McDonald (2008).
[20] Vgl. Microsoft (2022).

Es existieren verschiedene Organisationsformen für Cloud-Umgebungen[21]:

- **Public Cloud** Bei einer öffentlichen Cloud haben alle Menschen Zugang zu den verschiedensten Computer-Infrastrukturen über das Internet. Die Anbieter vermieten diese an die Kunden und führen die Abrechnung anhand der Nutzung durch. Die Kunden besitzen keinerlei Hardware selbst.
- **Private Cloud** Der Betrieb einer Private Cloud erfolgt einzig und allein für ein Unternehmen oder eine Organisation. Er kann sowohl intern, in einem eigenen Rechenzentrum, oder extern, durch Dritte, erfolgen. Ausschließlich das Unternehmen selbst hat Zugriff auf diese Cloud Umgebung.
- **Hybrid Cloud** Bei dem Ansatz einer hybriden Cloud wird der Zugang zu unterschiedlichen Infrastrukturen kombiniert. Zum Beispiel kann ein Teil der Ressourcen aus dem eigenen Rechenzentrum genutzt werden und ein Teil der Infrastrukturen eines öffentlichen Anbieters.
- **Community Cloud** Dieses Modell ähnelt dem der Public Cloud, aber der Zugang zu den Infrastrukturen ist nur für einen abgegrenzten kleineren Kreis an Nutzern vorgesehen, beispielsweise städtische Behörden oder Forschungseinrichtungen, wie Universitäten oder Krankenhäuser.

Die folgende Abbildung 1 zeigt die verschiedenen Arten von Cloud-Umgebungen.

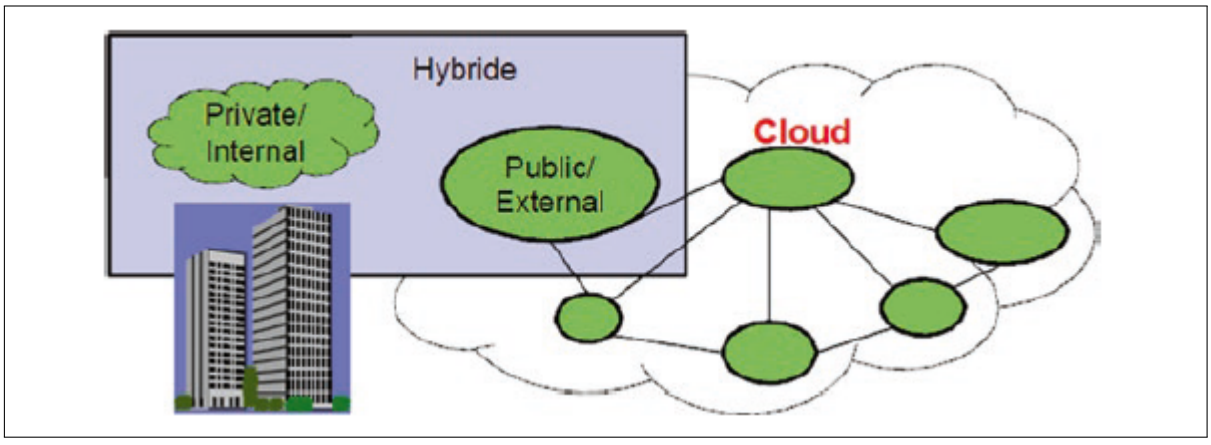


Abbildung 1: Verschiedene Cloud-Typen (Quelle: Luntovskyy, A./Gütter, D. (2020))

5.2 Container

Bei der Containervirtualisierung wird der Kernel des Hostsystems geteilt genutzt, durch unterschiedliche Instanzen von Betriebssystemen. Im Vergleich zur herkömmlichen Vir-

[21] Vgl. Luntovskyy, A./Gütter, D. (2020), 379f.

tualisierung unter Nutzung eines Hypervisors bietet die Containervirtualisierung einige Vorteile, aber auch Nachteile.

Eine Gemeinsamkeit beider Technologien ist die Isolation von einer Anwendung gegenüber der Umgebung, sodass diese isolierte Einheit zwischen unterschiedlichen Hostsystemen bewegt werden kann. Der Unterschied zwischen beiden Ansätzen liegt in der verwendeten Architektur. Die Architektur wird in der folgenden Abbildung 2 verdeutlicht.

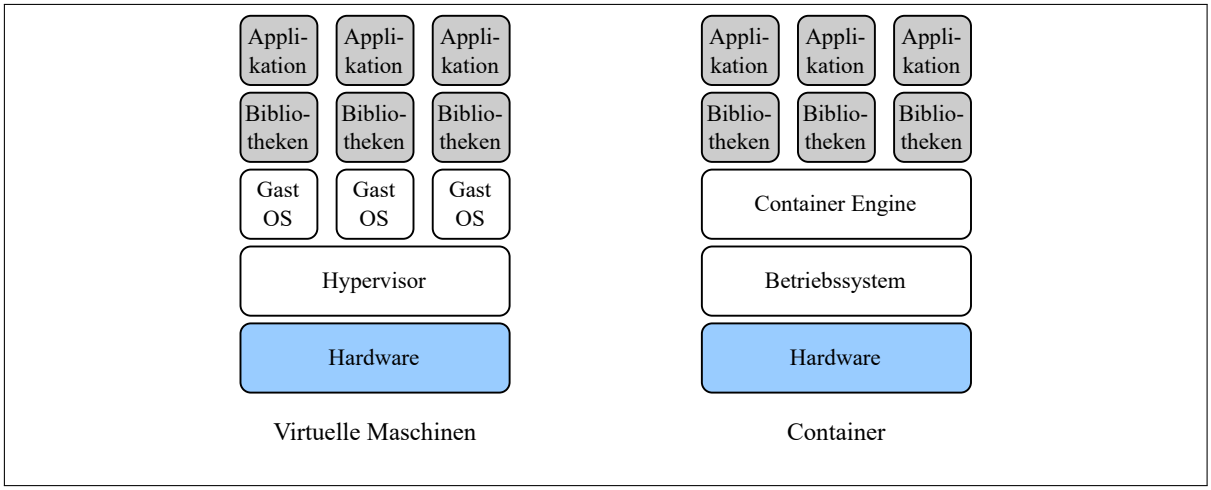


Abbildung 2: Architektur: Virtuelle Maschinen und Container (Quelle: eigene Darstellung)

Virtuelle Maschinen benötigen einen Hypervisor, welcher die Ausführung der Maschinen steuert und überwacht. Er kann entweder direkt auf der Hardware oder auch erst auf dem Betriebssystem installiert sein. Bei dieser klassischer Virtualisierung werden Hardware-Komponenten virtualisiert. Normalerweise kann ein Programm in einem herkömmlichen Betriebssystem alle Systemressourcen sehen und verwenden, wie beispielsweise die CPU, das Netzwerk oder angeschlossene Geräte. Der Zugriff auf diese Ressourcen kann eingeschränkt werden, je nachdem welcher Benutzer einen Prozess ausführt.

Die Containervirtualisierung geschieht auf Betriebssystem-Ebene. Es wird verwaltet, welche Ressourcen des Systems den Prozessen innerhalb eines Containers zugeordnet werden und genutzt werden können. Die Abstraktion auf Ebene des Betriebssystems geschieht mittels der Techniken *cgroups* und *namespaces* und dadurch sind Container flexibler einsetzbar, als virtuelle Maschinen. Container bieten die Möglichkeit zur abgeschotteten Serviceausführung. „Der Vorteil dieser Technik liegt in einer guten Integration von OS-Container und Gast-OS.“[22]

Alle Container auf einer Maschine nutzen den gleichen Kernel, wohingegen jede virtuelle Maschine ihren eigenen Kernel hat. „Systembeispiele für OS-Container sind Open Sola-

[22] Luntovskyy, A./Gütter, D. (2020), S. 391.

ris Zoning, BSD jails, OpenVZ, Virtuozzo, Linux-VServer“[23] oder die wohl bekannteste Umsetzung Docker.

Im Folgendem wird kurz auf Docker eingegangen. Docker ist im Jahr 2013 erschienen, in der Programmiersprache Go geschrieben. Docker ist „... ein Computerprogramm, das eine sogenannte ‚Sandbox‘ (Container) bereitstellt, die es ermöglicht, Programme/Apps isoliert (virtualisiert) vom BS auszuführen.“[24] Es nutzt die beiden bereits erwähnten Techniken *cgroups* und *namespaces* und als API-Schnittstelle zum Linux-Kernel die Eigenentwicklung *libcontainer*. Die *cgroups* übernehmen, vereinfacht dargestellt, die Verwaltung der Menge an Hardware-Ressourcen, die ein Container verwenden darf. Dadurch erfolgt eine Überwachung und Beschränkung der Ressourcen. Die *namespaces* regeln, was ein Container sehen kann und was nicht. Für einen Container, genauer gesagt den Prozess, scheint es, dass er allein läuft, in der Realität teilt er sich seine Ressourcen mit allen anderen Prozessen. Docker ist auf die Verwendung auf Linux-Systemen ausgerichtet, kann aber auch auf anderen Betriebssystemen genutzt werden.

Im Zusammenhang mit Docker werden einige Begriffe häufig verwendet, diese werden nachfolgend kurz erläutert:

- **Image** Ein Image ist ein Abbild eines Containers und besteht aus verschiedenen Layern, diese sind schreibgeschützt und dadurch unveränderlich. Ein Image kann stets für mehrere Container verwendet werden und ist portabel. Es wird meistens in einer Registry gespeichert.
- **Container** Container stellen die aktiv ausgeführte Instanz von einem Image dar. Führt ein Container kein Programm aus oder ist fertig mit der Arbeit, wird er automatisch wieder beendet.
- **Layer** Ein Image besteht aus vielen Layern, diese enthalten entweder eine Datei oder einen Befehl. Sie werden dann zum Image hinzugefügt. Mittels der Layer kann der Verlauf und der Aufbau eines Images untersucht werden.
- **Dockerfile** Das Dockerfile ist eine Textdatei. kann als eine Art Bauanleitung für ein Image gesehen werden. Während der Ausführung werden die einzelnen Schritte nacheinander abgearbeitet und jeweils ein Layer erzeugt.
- **Registry** Innerhalb einer Registry werden die einzelnen Images, mit mehreren verschiedenen Versionen, gespeichert. Diese können öffentlich zugänglich sein oder in Eigenverantwortung betrieben werden. Ein Beispiel wäre der Docker Hub. Zusätzlich können Images in der Registry gescannt und überprüft werden.

[23] Luntovskyy, A./Gütter, D. (2020), S. 391.
[24] ebenda, S. 393.

Änderungen an einem Docker Image selbst sind nicht vorgesehen. Soll eine Datei innerhalb eines Images verändert werden, so geschieht dies durch ein Dockerfile. In diesem werden die nötigen Anweisungen gegeben, daraufhin werden bei der Ausführung dieses Dockerfiles die einzelnen Anweisungen zu den Layern. So wird die veränderte Datei im Image nur während der Laufzeit des Containers genutzt. Das Image bleibt während der Laufzeit und auch danach unverändert.

Container bieten einen riesigen Vorteil in der Entwicklung von neuen Anwendungen oder der Erweiterung von Bestehenden. Durch die Einfachheit der Images und den Dockerfiles, als „Bauanleitungen“, kann ein Entwickler sich voll und ganz auf die eigentliche Anwendung konzentrieren. Der klare Aufbau, mit definierten Schnittstellen nach außen, ermöglicht auch eine vereinfachte Fehlersuche. Ein Docker Image verhält sich jederzeit gleich.

In den allermeisten Fällen werden Daten nicht innerhalb eines Containers gespeichert, sondern werden außerhalb auf persistenten Speichermedien oder in Datenbanken abgelegt. Dieser Fall beruht auf der Kurzlebigkeit der Container, sollte ein Container während seiner Laufzeit kaputt gehen, sind die Daten weiterhin gesichert, da diese nur von extern geladen wurden sind.

5.3 Micro-Services

Bisher sind nahezu alle Anwendungen große, monolithische Instanzen, welche meist als nur ein einziger Prozess oder eine kleine Anzahl davon auf wenigen Servern gleichzeitig laufen. Eine solche Anwendung besteht aus verschiedenen Komponenten, welche wiederum eng miteinander gekoppelt sind. Sie werden als Gesamtheit entwickelt, verwaltet und bereitgestellt. Wird ein Teil dieser Anwendung verändert, so muss die gesamte Anwendung erneut bereitgestellt werden. Durch das Nichtvorhandensein von Grenzen zwischen einzelnen Komponenten steigt die Komplexität des Gesamtsystems und damit unter Umständen einhergehend sinkt die Qualität.

Im Normalfall erfordert das Ausführen einer solchen monolithischen Anwendung viel Ressourcen und damit einen stark ausgestatteten Server. Wächst die Belastung des Systems, gibt es zwei verschiedene Wege es zu skalieren, entweder vertikal oder horizontal. Bei der vertikalen Skalierung werden mehr Ressourcen, wie Prozessoren oder Arbeitsspeicher hinzugefügt. Dieses Vorgehen wird schnell sehr teuer und ist meist durch eine technische Obergrenze limitiert. Bei der horizontalen Skalierung werden mehrere Server verwendet, mit je einer eigenen Instanz. Bei diesem Vorgehen sind Änderungen am Programmcode selbst notwendig. Nicht immer ist diese Skalierung möglich, beispielsweise bei relationalen Datenbanken. Ist die Skalierung einer Komponente nicht möglich, so trifft dies auch meist

auf die gesamte Anwendung zu, außer eine Aufteilung der Anwendung ist möglich. Meist hatten solche Anwendungen einen großen Release-Lebenszyklus und erhielten selten eine Aktualisierung. Die Entwickler übergaben die fertige Anwendung am Ende des Release-Zyklus an das entsprechende Betriebsteam, welches die Software dann installierte und kontrollierte.

Diese und weitere Probleme von monolithischen Anwendungen haben dazu geführt, Anwendungen in unabhängige und kleinere Komponenten zu unterteilen, ihre Bezeichnung lautet Micro-Services. „Die Micro-Service-Architektur wird über alle einzelnen Micro-Services hinweg als große Einheit dargestellt."[25] Sam Newman beschäftigte sich umfassend mit Micro-Services und veröffentlichte zwei, oft referenzierte, Bücher zu diesem Thema, so schrieb er: „Micro-Service is a self-executable software component that collaborates with other software components within an integrated application. Micro-Services should take on a role within a specialized business domain."[26] Jede solche Komponente wird als ein eigenständiger Prozess ausgeführt. Micro-Services kommunizieren untereinander über klar definierte und verständliche Schnittstellen. Es können sowohl synchrone als auch asynchrone Protokolle genutzt werden. Weitverbreitet ist die Nutzung von Hypertext Transfer Protocol (HTTP) und einer Representational State Transfer (REST)-fähigen API. Dadurch das jeder Micro-Service ein eigener Prozess mit extern nutzbarer API ist, werden sie getrennt voneinander entwickelt und zur Verfügung gestellt, sodass in Folge einer Änderung keine neue Bereitstellung von anderen Micro-Services erforderlich ist, unter der Prämisse, dass die API maximal in einer abwärtskompatiblen Art und Weise verändert wurde. Durch den Ausfall eines solchen Micro-Services kommt es nicht direkt zu einem Ausfall des Gesamtsystems.

Die Abbildung 3 vergleicht die Architektur einer monolithischen Anwendung mit der einer Anwendung, welche aus Micro-Services besteht.

[25] Luntovskyy, A. (2021b), S. 428.
[26] Newman, S. (2019), 1f.

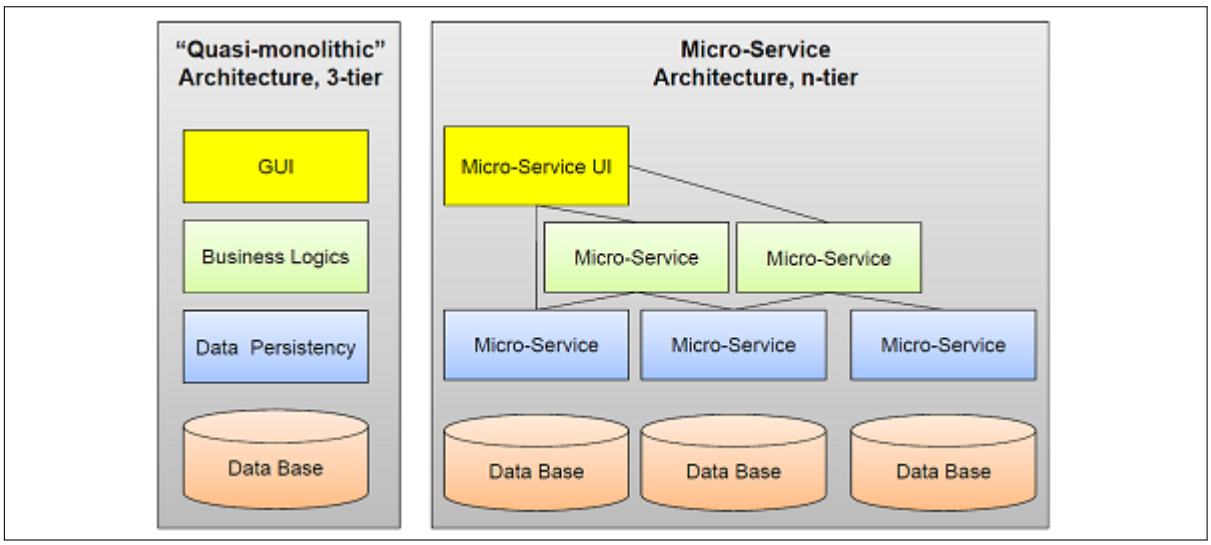


Abbildung 3: Vergleich Architektur: Monolithische Anwendung und Micro-Services (Quelle: Luntovskyy, A. (2021b))

Durch die klare Trennung und Unterteilung der Komponenten wird die Möglichkeit der Skalierung deutlich verbessert, sodass nur die Micro-Services skaliert werden müssen, welche tatsächlich stärker belastet werden. Je nach Möglichkeit können so bestimmte Komponenten horizontal oder vertikal skaliert werden.

Durch die steigende Anzahl einzelner Komponenten steigt auch der Aufwand der Konfiguration und Verwaltung, da sichergestellt werden muss, dass alle Teile miteinander kommunizieren können. In Projekten mit wenigen einzelnen Micro-Services ist dies noch überschaubar, dennoch steigt mit zunehmender Anzahl die Fehleranfälligkeit.

Ein weiterer Fakt, welcher bei Micro-Services berücksichtigt werden muss, ist das erschwerte Debuggen von Funktionsaufrufen, da diese meist mehrere Computer und Prozesse umfassen. Moderne Implementierungen bieten dafür Lösungsansätze.

Es findet eine Unterscheidung in zustandslose und zustandsbehaftete Micro-Services statt. Zustandslose Micro-Services liefern bei gleichem Input stets den gleichen Output. Hingegen beim zustandsbehafteten ist der Output abhängig vom aktuellen Zustand des Services. Dieser Umstand muss bei der Wiederherstellung eines Micro-Services nach dessen Ausfall berücksichtigt werden.

Es werden gewisse Software Implementierungen und Umsetzungen benötigt, um Micro-Services zu verwalten. „The main platforms and frameworks for ... Micro-Services are as follows: ... Spring, Kubernetes.“[27] Im folgendem Kapitel wird unter anderem Kubernetes zur Verwaltung dargestellt.

[27] Klymash, M./Beshley, M./Luntovskyy, A. (2022), S. 26.

5.4 Containerorchestrierung

Vor allem in service-orientierten Architekturen ist die zentrale Orchestrierung essentiell. Unter Orchestrierung versteht man die automatische Konfiguration, Koordinierung und Verwaltung von Computer Systemen und/ oder Software. Ziel ist die Infrastruktur, Anwendungen oder Daten so anzuordnen, dass das Ziel bestmöglich erreicht wird. Orchestrierung umfasst unter anderem auch bestehende Prozessautomatisierungen. Durch die Angabe bestimmter Parameter und Grenzwerte, können Quality of Service (QoS) oder minimale Kosten erreicht werden, beispielsweise durch die Bereitstellung weiterer Ressourcen, wenn die Last eines Systems steigt. „QoS beschreibt die Güte eines Kommunikationsdienstes aus der Sicht der Anwender (Datenrate, Latenz, Jitter, Verlustrate etc.).“[28]

Eine Anwendung besteht selten aus nur einem Container, sondern ist ein Zusammenspiel von mehreren einzelnen, wie zum Beispiel einem Container für den Webserver, einem für die Datenbank und für Anwendung selbst einem. Um mehrere Container gleichzeitig zu verwalten und zu überwachen, ist Containerorchestrierung notwendig, dadurch kann das gewünschte Verhalten der Gesamtanwendung sichergestellt werden.

Im Folgendem werden zwei mögliche Umsetzungen der Containerorchestrierung dargestellt.

Die Erste ist die in Docker selbst enthaltene Umsetzung Swarm. Es ist nativ in der Docker-Engine enthalten. Mehrere Host-Systeme mit installiertem Docker können mittels Swarm zusammengefasst werden und dadurch zentral verwaltet werden. Es wird eine Master-Slave-Architektur genutzt, sodass es mindestens einen Manager zur Verwaltung des Clusters gibt und beliebig viele Worker zur Ausführung der eigentlichen Arbeit. Die Container-Anwendungen werden als Services bezeichnet und auf den Knoten verteilt. Swarm enthält eine integrierte Lastausgleichsfunktion, sodass eingehende Anfragen automatisch und selbstständig an alle dafür zur Verfügung stehenden Services verteilt werden. Die Verwaltung eines Swarm Clusters geschieht durch das bekannte Docker Kommandozeilenwerkzeug.[29]

Die zweite Umsetzung von Containerorchestrierung ist Kubernetes. Es wurde durch Google entworfen und entwickelt, aber das Projekt wurde nach einem Jahr an die Cloud Native Computing Foundation (CNCF) gespendet. Es ist vollständig Open-Source. Ziel ist die Automatisierung der Prozesse, um Container-Anwendungen bereitzustellen, zu skalieren und zu verwalten. Kubernetes unterstützt verschiedene Werkzeuge und Plattformen für Container, wie beispielsweise Docker. Kommerzielle Cloud-Anbieter, wie Amazon oder

[28] Luntovskyy, A./Gütter, D. (2020), S. 477.
[29] Vgl. Docker Inc. (2021).

Microsoft, verwenden Kubernetes, um Kunden Cluster anzubieten, diese werden durch den Anbieter installiert und konfiguriert, sodass der Kunde nur noch seine Anwendungen darin ausführen muss und somit der Verwaltungsaufwand erheblich gesenkt wird.

Kubernetes bietet folgende Funktionen[30]:

- **Service Discovery und Load Balancing** Kubernetes bietet die Möglichkeit Containern DNS-Namen zu zuordnen. Um die Bereitstellung einer Anwendung stabil zuhalten, kann Kubernetes automatisch die Last verteilen.
- **Speicherorchestrierung** Kubernetes bietet die Möglichkeit verschiedenste Speicher einzubinden, egal ob lokal oder extern von Cloud-Anbietern. Eine Anwendung innerhalb eines Containers benötigt keine Veränderung, da diese die Schnittstelle von Kubernetes zu Kommunikation nutzen.
- **Automatische Rollbacks und Rollouts** Es muss nur der fertige Zustand der Container beschrieben werden. Kubernetes übernimmt die nötigen Schritte und Anpassungen, um dieses Ziel zu erreichen.
- **Automatisches Bin Packing** Kubernetes weiß wie viel Ressourcen ein Container benötigt und wird diese dann so auf die Knoten des Clusters verteilen, dass die Ressourcen effizient und optimal genutzt werden.
- **Selbstheilung** Fällt ein Container aus, so startet Kubernetes diesen neu. Durch den Anwender können Überprüfungen definiert werden, reagiert ein Container nicht auf diese, wird er gestoppt. Container sind erst für Clients zugänglich, wenn sie arbeitsfähig und bereit sind.
- **Verwaltung von Geheimnissen und Konfigurationen** Vertrauliche Daten können innerhalb von Kubernetes gespeichert und verwaltet werden, sodass Container Images nicht neu erstellt werden müssen.

Kubernetes verwendet auch eine Master-Slave-Architektur. Der Master steuert durch seine Komponenten die einzelnen Nodes. Auf Letzterem laufen die eigentlichen Container in den *Pods*, sie sind die kleinste nutzbare Einheit. Sie bestehen aus einem einzelnen oder mehreren Containern. Diese teilen sich die zugeordneten Ressourcen und die Container-Runtime, damit sind die kleinsten Komponenten, welche zur Verteilung der Last genutzt werden kann.

[30] Vgl. The Kubernetes Authors (2022).

6 Implementierungskonzepte für Confidential Computing

Die Möglichkeit vertrauliche Daten mit vertrauenswürdigen, meist eigenen, Code in einer nicht vertrauten Umgebung zu verarbeiten, wird als Sicheres Rechnen auf entfernten Rechnern (engl.: *Secure Remote Computation (SRC)*) bezeichnet. Dabei wird weder der Hardware, dem Betriebssystem noch den Administratoren vertraut. Das Ziel dieses Sicheren Rechnens auf entfernten Rechnern ist die Einhaltung des Schutzes der Vertraulichkeit und Integrität. Das System, welches zur Verarbeitung genutzt wird, steht dabei meistens nicht unter Kontrolle des Anwenders, sondern wird durch eine dritte Partei verwaltet und zur Verfügung gestellt. SRC ist eine der größten Herausforderungen seit vielen Jahren im Rahmen der IT-Sicherheit.[31] Ein bisheriger Lösungsansatz für das Problem des Sicheren Rechnens auf entfernten Rechnern ist die Homomorphe Verschlüsselung, beispielsweise das Verfahren von Gentry.[32] Dabei werden Funktionen auf verschlüsselte Daten angewendet, sodass weder die Eingabe- noch die Ausgabedaten zu keinem Zeitpunkt entschlüsselt vorliegen. Durch bestimmte mathematische Eigenschaften liegt das richtige Ergebnis nach der abschließenden Entschlüsselung vor.[33]

Die homomorphe Verschlüsselung findet praktisch nur sehr wenig Einsatz, da diese einen erheblichen Einfluss auf die Performanz hat.[34]

Ein weiterer Weg Daten gesichert auf nicht vertrauenswürdigen Systemen zu verarbeiten ist die Verwendung von besonderer Hardware im entfernten System. Diese spezielle Umgebung wird als Trusted Execution Environment (TEE)[35] bezeichnet. Sie dient der Wahrung der Integrität und Vertraulichkeit der Daten. Das Vertrauen beruht nur auf der Funktionsweise dieser speziellen Komponente und damit dessen Hersteller. Es muss nicht der gesamten Rechenumgebung vertraut werden.

6.1 Intel SGX

Intel, als Prozessorhersteller, verwendet TEE seit dem Jahr 2015 in seinen Prozessoren, erstmals in der Skylake Generation. Intel nennt diese Technologie Software Guard Extensions (SGX). Bereits zwei Jahre zuvor wurde diese Erweiterung des Prozessorbefehlssatzes angekündigt und der Öffentlichkeit gezeigt.[36] SGX führt eine Isolierung auf der zugrun-

[31] Vgl. Costan, V./Lebedev, I./Devadas, S. (2017), S. 2 f.
[32] Vgl. Gentry, C. (2009).
[33] Vgl. ebenda, S. 5.
[34] Vgl. Naehrig, M./Lauter, K./Vaikuntanathan, V. (2011).
[35] Vgl. Sabt, M./Achemlal, M./Bouabdallah, A. (2015).
[36] Vgl. McKeen, F. et al. (2013).

deliegenden Hardwareebene durch, der vertrauenswürdige Code wird isoliert behandelt, sodass nicht berechtigte Zugriffe verhindert werden, während der Berechnungen.[37]

6.1.1 Bedrohung

Um die Entwicklung der Technologie Intel SGX nachvollziehen zu können, muss das Bedrohungsmodell verstanden werden, welches zur Konzeptionierung verwendet wurde.

Die wichtigsten Schutzziele in diesem Modell sind die Vertraulichkeit und Integrität der sensitiven Daten, welche durch eine Anwendung verarbeitet werden. Angriffe darauf sowie auf den eigentlichen Programmcode zur Verarbeitung sollen verhindert werden. Die Verfügbarkeit als Schutzziel steht nicht im Fokus dieser Technologie.

Dieses Modell schreibt dem Angreifer weitgehende Möglichkeiten zu, um Zugriff auf die vertrauenswürdigen Daten zu erlangen. Die Umgebung in welcher eine Anwendung ausgeführt wird, kann der Angreifer wie folgt verändern und manipulieren:

- Die zugrundeliegende Hardware mit allen dazugehörigen Ein- und Ausgängen, die CPU selbst ist davon aber ausgenommen.
- Das gesamte verwendete Betriebssystem und der zugrundeliegende Kernel, um beispielsweise Veränderungen am Hauptspeicher vorzunehmen.
- Alle Benutzerkonten, einschließlich privilegierter, zum Beispiel Administratoren.
- Alle anderen Anwendungen und Prozesse.
- Die Bestandteile innerhalb des Gesamtprogramms, welche außerhalb der abgesicherten SGX-Umgebung stattfinden und vorhanden sind.

Durch diese weitgehenden Möglichkeiten wird nur SGX selbst innerhalb der Central Processing Unit (CPU) als sicher angesehen, sowie das Vertrauen darin gelegt. Die abgesicherte Anwendung wird auch als sicher betrachtet und kann daher zur Verarbeitung genutzt werden.[38]

6.1.2 Funktionsweise

Um die Trennung von vertrauenswürdigen Code und Daten zur nicht vertrauten Umgebung zu realisieren, verwendet die SGX Technologie sogenannte Enklaven (engl.: *Secure*

[37] Vgl. Costan, V./Lebedev, I./Devadas, S. (2017), S. 3 f.
[38] Vgl. Priebe, C. et al. (2019).

Enclaves) um den abzusichernden Code zu schützen. Enklaven stellen private, durch die CPU abgeschirmte, Speicherbereiche zur Verfügung, welche exklusiv nur von der Enklave genutzt werden können.

Die Zuteilung und Verwaltung der Speicherbereiche geschieht auf der Ebene der Hardware. Die Zugriffsregeln werden ebenda verwaltet und kontrolliert, sodass weder ein Administrator, eine Anwendung noch das Betriebssystem samt Treibern selbst auf den Speicher, welcher einer Enklave zugeteilt wurden ist, zugreifen können. Dieser zugeteilte Speicher wird als Enclave Page Cache (EPC) bezeichnet.[39]

Eigenschaften Folgende Eigenschaften werden durch die Software Guard Extensions zur Verfügung gestellt und sind sicherheitsrelevant. d Diese stammen aus den *SGX Explained*[40] und *SGX Bootstrap*[41] Veröffentlichungen:

- Anwendungen können in einem von vier verschiedenen Modi ausgeführt werden: *Production, Pre-release, Debug, Simulation.*
- Im *Production* Modus kann kein Debugger zur Untersuchung genutzt werden.
- Im *Simulation* Modus werden keine echten SGX-Enklaven genutzt.
- Jede Enklave kann nur auf ihren zugeordneten Speicher zugreifen. Der Zugriff kann nur für sie selbst erfolgen.
- Der Zugriff auf Enklaven, beispielsweise durch Funktionsaufrufe ist nur möglich, wenn bestimmte Rahmenbedingungen eingehalten werden.
- Enklaven können keine Systemcalls durchführen oder auf Interrupts reagieren, um keine Daten preiszugeben.
- Daten, welche eine Enklaven verlassen, werden durch den Prozessor verschlüsselt.
- Die CPU erhält während der Herstellung *Root Keys*, aus diesen können kryptografische Schlüssel abgeleitet werden. Sie sind nicht auslesbar.
- Das Prinzip der Attestation kann verwendet werden, um die Integrität des auszuführenden Codes innerhalb der Enklave sicherzustellen.

Eine Anwendung, welche die SGX-Technologie nutzt und sensitive Daten verarbeiten soll, muss aus zwei verschiedenen Teilen bestehen um die Integrität und Vertraulichkeit durch Enklaven zu schützen. Der Aufbau wird in der Abbildung 4 gezeigt.

[39] Vgl. Costan, V./Devadas, S. (2016), S. 2.
[40] Vgl. ebenda.
[41] Vgl. Georgia Institute of Technology (2019a).

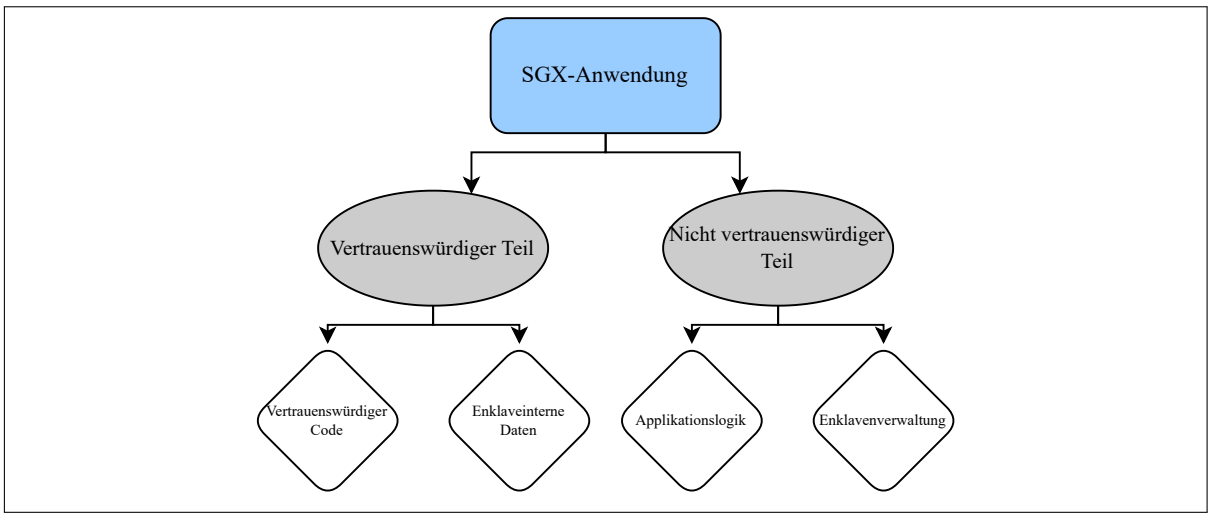


Abbildung 4: Aufbau einer SGX-Anwendung (Quelle: eigene Darstellung)

Der erste Teil ist die vertrauenswürdige Komponente der Anwendung. Dieser Teil kann auf die Schlüssel zugreifen und die Ausführung dieses Teiles findet in der Enklave statt. Diese Komponente enthält den Programmcode zur Ausführung in der Enklave sowie alle internen Daten der Enklave selbst.

Im zweiten Teil der Anwendung befinden sich alle nicht vertrauenswürdigen Bestandteile. Dieser Teil wird ganz normal als Systemprozess zur Ausführung gebracht. Er ist zuständig für das Starten des vertrauenswürdigen Teils in einer Enklave, unter Umständen können auch mehrere Enklaven genutzt werden. Des Weiteren umfasst diese Komponente jegliche sonstige Applikationslogik, welche keinen weiteren Schutz bedarf.[42]

Die Unterteilung in diese beiden Teile sollte bereits zu Anfang der Entwicklung einer neuen Anwendung große Bedeutung haben und beachtet werden. Bei bereits vorhandenen Anwendungen ist die Partitionierung deutlich komplexer und aufwändiger, da zunächst die einzelnen Bestandteile der Anwendung analysiert werden müssen. Es muss evaluiert werden, welche Teile abgesichert ausgeführt werden. Zusätzlich muss der vorhanden Quellcode angepasst werden und es muss einen ausgiebigen Testlauf geben, um Probleme zu eliminieren, welche beispielsweise durch die Ausführung in einer Enklave entstehen könnten.

Die vertrauenswürdige Komponente der Anwendung wird im gleichen Prozess zur Ausführung gebracht, wie die Anwendung selbst. Dadurch kann sie auf den gesamten Speicher der Anwendung zugreifen, aber nicht die Anwendung auf den Speicher des vertrauenswürdigen Teils.

[42] Vgl. Adamski, A. (2018a).

Die Technologie beinhaltet kein Limit zur Anzahl an Enklaven innerhalb einer Anwendung, sodass der vertrauenswürdige Teil mehrere Enklaven enthalten kann, welche gleichberechtigt sind.[43]

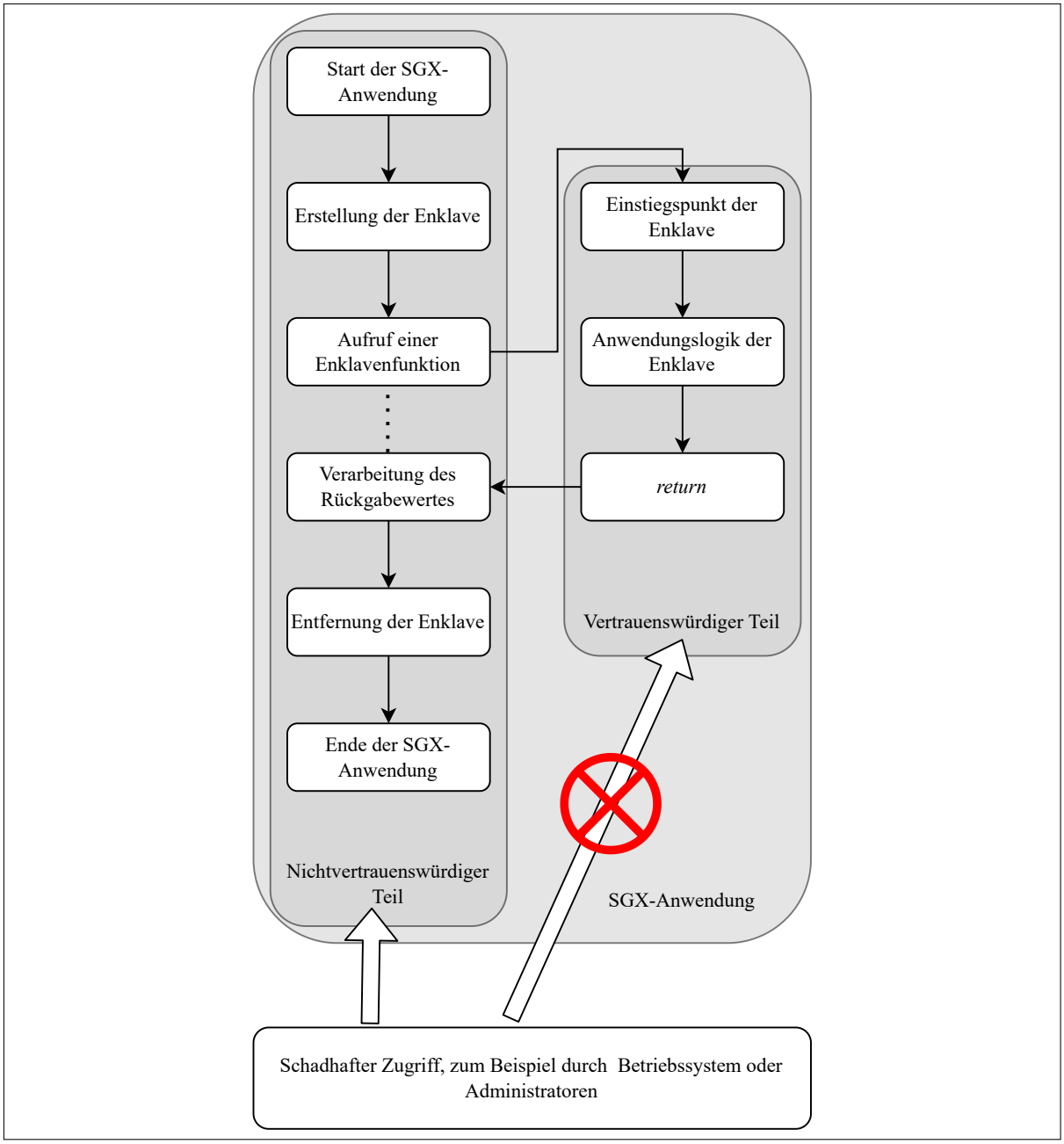


Abbildung 5: Ablauf einer SGX-Anwendung (Quelle: eigene Darstellung)

Ablauf einer SGX Anwendung In der 5 wird der schematische Ablauf einer SGX-Anwendung dargestellt. Die Anwendung muss zunächst die Enklave vorbereiten, dies geschieht mithilfe des Prozessors und befindet sich im nicht vertrauenswürdigen Teil der Anwendung, meist in der *main()* oder einer ähnlichen Funktion.[44]

[43] Vgl. Intel Corporation (2020a), S. 8.
[44] Vgl. Mechalas, J. P./Odom, B. J. (2016).

Es können weitere Schritte im Programm folgen, welche beispielsweise Vorberechnungen umfassen. An einer beliebigen Stelle im nicht vertrauenswürdigen Code kann dann die Enklave initialisiert werden, dabei bereitet die CPU den Code der Enklave sowie alle Informationen, welche für SGX benötigt werden, vor. Nach der *SGX Explained*[45] Veröffentlichung kann der Prozess durch drei Schritte beschrieben werden:

1. Benötigter Speicher wird im Enclave Page Cache reserviert und vorbereitet.
2. Der vertrauenswürdige Code und die enklaveninternen Daten werden in den EPC geladen. Daraufhin wird die Checksumme der Enklave aktualisiert, um den gesamten Speicherinhalt abzudecken.
3. Die Initialisierung wird beendet, außerdem wird die Autoren- und Enklavensignatur verifiziert.

Nach der erfolgreichen Erstellung der Enklave, kann an freigewählter Stelle der Anwendung eine Enklavenfunktion genutzt werden. Diese muss im Vorhinein definiert werden. Nach dem Aufruf dieser Funktion wird die Kontrolle an den vertrauenswürdigen Code innerhalb der Enklave durch den Prozessor übergeben. Die CPU übernimmt dabei Sicherheitsüberprüfungen. Voraussetzung für den Zugriff auf den Enclave Page Cache ist, dass der logische Prozessor im Betriebsmodus *Enclave Mode* ist. Durch die Ausführung der Enklavenfunktion wird die CPU in den entsprechenden Modus versetzt.

Nachdem der vertrauenswürdige Code in der Enklave ausgeführt wurden ist, verlässt die CPU den *Enclave Mode* und die Enklave selbst wieder. Der restliche Teil der Anwendung, welcher nicht vertrauenswürdig ist, wird nun weiter ausgeführt. Dabei können Rückgabewerte von der Enklave verwendet werden, sollte dies innerhalb der Anwendung gewünscht sein.[46]

Kommt es zu einem Hardware-Interrupt oder zu Fehlern beim Speicherzugriff während die CPU im Enclave Mode ist, verwendet sie einen Asynchronous Enclave Exit (AEX), um asynchron aus der Enklave auszusteigen. Dabei wird die aktuelle Ausführungsumgebung samt Registerzuständen gesichert. Zusätzlich werden Eigenschaften der Enklave abgelegt. Alle so entstandenen Daten sind verschlüsselt und der Prozessor löscht die zugehörigen Daten der Enklave, um keinerlei private Daten preiszugeben. Nach diesem Ausstieg übernimmt die Interrupt Service Routine (ISR) des verwendeten Betriebssystems. Der nicht vertrauenswürdige Teil der Anwendung kann daraufhin den vertrauenswürdigen Programmcode erneut ausführen, indem dieser die Enklave erneut betritt.

[45] Vgl. Costan, V./Devadas, S. (2016), S. 63 f.
[46] Vgl. ebenda, S. 65 ff.

Wird die Enklave innerhalb der SGX-Anwendung nicht mehr benötigt, veranlasst diese im nicht vertrauenswürdigen Teil die Terminierung und Löschung der Enklave. Der Prozessor löscht in diesem Fall alle Daten der Enklave und gibt die zugeordneten EPC-Speicherbereiche wieder frei. Danach werden die Metadaten entfernt und damit wird die Enklave endgültig terminiert. Die Anwendung selbst kann daraufhin, im nicht vertrauenswürdigen Teil, weiteren Code ausführen.[47]

6.1.3 Entwurf

Bei dem Entwurf von SGX-Anwendungen muss zunächst unterschieden werden, ob eine bestehende Anwendung um die Möglichkeit SGX-Enklaven zu nutzen erweitert werden soll oder ob eine neue Anwendung entwickelt wird, mit dem Ziel die SGX-Funktionalität zu verwenden. Bei Letzterem können die Besonderheiten bei der Entwicklung von Anfang an berücksichtigt werden. Bestehende Anwendungen können hingegen nicht ohne Veränderung die SGX-Technologien nutzen, da zunächst die Verwaltung der Enklaven hinzugefügt werden muss, zudem muss untersucht werden, welcher Code selbst in der Enklave ausgeführt werden soll. Für die Entwicklung einer neuen SGX-Anwendung hat Intel offizielle Informationen bereitgestellt im: *Intel Software Guard Extensions Tutorial*.[48] Dieses Vorgehen soll im folgendem beschrieben werden.

Partitionierung der Anwendung Zunächst müssen die formalen Anforderungen der SGX-Anwendung spezifiziert werden, danach kann die Unterscheidung in die Teile vertrauenswürdig und nicht vertrauenswürdig der Anwendung vorgenommen werden, um anhand dieser Verteilung die Anwendung selbst zu strukturieren. Nach erfolgter Strukturierung und Teilung können die Vorteile und damit die einhergehende Sicherheit der Enklave genutzt werden. Die Unterteilung lässt sich anhand der folgenden vier Schritte vornehmen:

1. Die zu schützenden Daten werden identifiziert.
2. Die Quellen und Verarbeiter dieser Daten werden abgeleitet. Es wird versucht die Zahl dieser zu senken.
3. Die Vertrauenswürdige Komponente der Anwendung wird spezifiziert.
4. Die Bestandteile zur Verwaltung und Kommunikation mit der Enklave werden entwickelt.

[47] Vgl. Costan, V./Devadas, S. (2016), S. 66 f.
[48] Vgl. Mechalas, J. P./Reed, I. (2016).

Durch die Verwendung dieses Vorgehens folgt man der Empfehlung des Entwicklerhandbuchs seitens Intel. Dies geschieht mit der Vorgabe, den Code, welcher innerhalb einer Enklave ausgeführt wird, so gering wie möglich zu halten. Dadurch sinkt die Angriffsfläche der Anwendung, was wiederum eine höhere Sicherheit mit sich bringt.[49] Dieses zur Ausführung des vertrauenswürdigen Codes genutzte Umfeld wird als Trusted Computing Base (TCB) bezeichnet. Durch die beschriebene Reduktion dessen wird insgesamt weniger Speicher im Enclave Page Cache benötigt, dadurch können mehr Enklaven gleichzeitig existieren, wenn die Gesamtgröße gleich bleibt. Außerdem werden weniger Auslagerungsvorgänge vom Enklavenspeicher in den Hauptspeicher benötigt, diese sind sehr zeitintensiv. Im Abschnitt 6.1.7 wird dies genauer beschrieben. Weiterhin umfasst die TCB diejenigen Soft- und Hardwarekomponenten eines Systems, welche als sicher angesehen werden und damit Vertrauen entgegen gebracht werden kann. Für den Fall von Intel SGX ist das der Prozessor, der Enclave Page Cache, die Implementierung von SGX und der vertrauenswürdige Anteil einer SGX-Anwendung.

Sind die zu schützenden Daten identifiziert, kann der vertrauenswürdige Anwendungsteil um diese herum entworfen werden, sodass der Programmcode innerhalb der Enklave wirklich nur der Verarbeitung dieser Daten gilt. In den nicht vertrauenswürdigen Teil der Anwendung können dann alle Operationen und Berechnungen ausgelagert werden, welche nicht von diesen Daten abhängen.

Die Begrenzung der Enklave sollte so gewählt sein, dass ein Kompromiss passend für den spezifischen Anwendungsfall gefunden wird aus den folgenden Zielen:

- Innerhalb der Enklave soll versucht werden die zu schützenden Daten nahezu vollständig zu verarbeiten.
- Die Größe des vertrauenswürdigen Programmcodes soll minimiert werden durch die Verwendung weniger Komponenten, welche diese Daten verarbeiten.
- Der Programmcode innerhalb der Enklave soll möglichst unabhängig und wenig Interaktionen mit dem nicht vertrauenswürdigem Programmcode haben.

Die Herausgabe von sensitiven Informationen aus der Enklave lässt sich nicht immer unterbinden, dies muss berücksichtigt werden. Enklaven können selbst keine Ein- oder Ausgabe-Operationen ausführen, sodass, für den Fall gewünscht, Informationen auf einem Bildschirm für den Nutzer ausgegeben werden sollen, diese an das Betriebssystem übergeben werden müssen.

[49] Vgl. Intel Corporation (2020a), S. 8.

Wird SGX hingegen in einer Cloud-Umgebung verwendet, kann das ohnehin schon verwendete Netzwerk zum Datenaustausch für Ein- und Ausgaben verwendet werden, da diese noch in der Enklave selbst mittels TLS ent- und verschlüsselt werden können.

Enklavenschnittstellen Im Abschnitt 6.1.2 wurde bereits gezeigt und erläutert, dass wenn auf eine Enklave innerhalb einer Anwendung zugegriffen werden soll, dies durch den nicht vertrauenswürdigen Teil geschieht. Dieser Teil ist notwendig für Ein- und Ausgaben, wie beispielsweise dem Einlesen von Eingabewerten, diese werden dann zur Enklave weitergeleitet. Wurden die Werte in der Enklave verarbeitet, nimmt dieser Teil auch wieder die Ergebnisse entgegen. Anschließend können diese ausgegeben werden.

Die Kommunikation zwischen den beiden Teilen einer Anwendung kann auf drei verschiedene Art und Weisen betrachtet werden:[50]

ECall: Wird eine Funktion innerhalb einer Enklave im nicht vertrauenswürdigen Code aufgerufen, spricht man vom *Enclave Call*, kurz: ECall. Der Prozess überreicht die verwendeten Funktionsargumente an die Enklave. Daraufhin wird die Anwendung pausiert und der Prozessor wechselt seinen Modus in den Enclave Mode.

OCall: Wurden die Berechnungen innerhalb der Enklave fertiggestellt, nutzt die Enklave den sogenannten *Outside Call*, kurz OCall, um die Kontrolle der Anwendung selbst zu übergeben, damit verlässt die CPU den Enclave Mode wieder. Die Ausführung der Anwendung läuft nach dem vorangegangen ECall weiter.

AEX: Durch den Asynchronous Enclave Exit wird bei einem Interrupt die Ausführung von vertrauenswürdigen Code innerhalb der Enklave beendet. Dies wurde bereits im Abschnitt 6.1.2 beschrieben.

Beim Entwurf der Enklavenschnittstellen zwischen dem misstrauten und dem vertrauten Teil der Anwendung muss davon ausgegangen werden, dass jegliche Komponenten und Codeteile, welche nicht in der Trusted Computing Base liegen, Informationen und Daten verfälschen, blockieren oder weitergeben können. Demzufolge dürfen bei einem Outside Call keinerlei sensitive Daten aus der Enklave getragen werden. Bei dem Enclave Call hingegen müssen die Parameter dessen innerhalb der Enklave durch den Programmcode überprüft werden.

[50] Vgl. Adamski, A. (2018b).

Der Datenaustausch mit der Enklave erfordert, dass die genutzten Funktionen bei der Entwicklung der Anwendung spezifiziert werden. Es werden sogenannte *Proxy Functions* oder *Bridge Functions* für die Übertragung von Informationen genutzt.[51] Intel hat ein Hilfsprogramm zur Entwicklung von SGX-Anwendungen und diesen Übertragungsfunktionen entwickelt. Das Programm *Edger8r* generiert automatisch für jeden ECall und OCall diese, jeweils eine für den Teil innerhalb der Enklave und eine für den nicht vertrauenswürdigen Teil außerhalb der Enklave.[52]

Wird eine Enklavenfunktion innerhalb einer SGX-Anwendung ausgeführt, so geschieht dies indirekt. Die durch Edger8r generierte Funktion wird aufgerufen, diese führt den Enclave Call durch und startet die vertrauenswürdige Funktion innerhalb der Enklave. Die Enklavenfunktion im vertrauenswürdigen Teil des Codes selbst wird dann dadurch aufgerufen und durchgeführt. Dieses Vorgehen wird invertiert bei dem Outside Call.[53]

Der Wechsel zwischen dem Enclave Mode und dem normalen Zustand der CPU erfordert eine korrekt vorbereitete Umgebung, sodass diese zusätzlichen Funktionen notwendig werden. Zum Beispiel müssen Funktionsargumente vom Hauptspeicher in den Enclave Page Cache geladen werden oder auch wieder zurück kopiert, außerdem ist eine Überprüfung auf bestimmte Bedingungen notwendig, wie beispielsweise die maximal zulässige Größe des Caches.

Zusätzlich kann auf diese Weise auf die Rückgabewerte der Enclave Call-/ Outside Call-Operationen innerhalb der Hilfsfunktionen sowie dem Rückgabewert der eigentlich aufgerufenen Funktion zugegriffen werden. Letzterer wird mittels eines Pointers referenziert.[54]

Die folgende Abbildung 6 verdeutlicht die Folge der Funktionsaufrufe bei ECalls und OCalls:

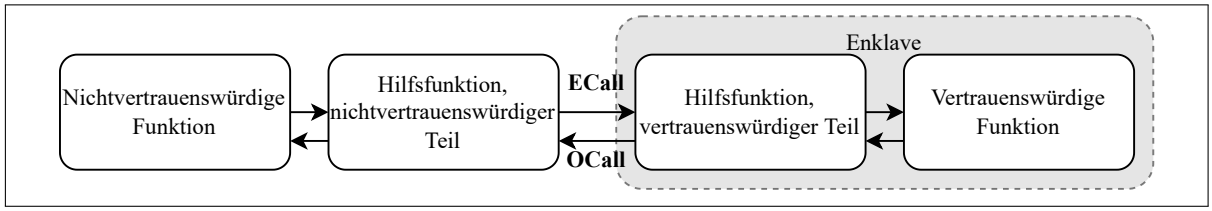


Abbildung 6: Folge der Funktionsaufrufe bei ECalls und OCalls (Quelle: eigene Darstellung)

[51] Vgl. Mechalas, J. P./Reed, I. (2016).
[52] Vgl. Intel Corporation (2020a), S. 14 ff.
[53] Vgl. Georgia Institute of Technology (2019b).
[54] Vgl. Mechalas, J. P. (2016).

6.1.4 SGX-interne Komponenten

Um die Funktionsweise von Intel SGX genauer zu verstehen und die gewonnenen Sicherheitsmerkmale einschätzen zu können, ist es notwendig bestimmte interne Komponenten zu kennen und einzuordnen. In diesem Abschnitt wird ein Überblick über die wesentlichsten Konzepte und dafür wichtigen Detailinformationen für Intel SGX-Enklaven geschaffen.

Zunächst wird auf den Enclave Page Cache, dem internen Hauptspeicher der Enklave, eingegangen, danach werden die kryptografischen Schlüssel zur Sicherung der Authentizität, Integrität und Vertraulichkeit thematisiert.

Enclave Page Cache Der Enclave Page Cache ist die zentrale Komponente bei der Umsetzung der Software Guard Extensions. Er ermöglicht, dass die zu schützenden Daten mithilfe des Prozessors isoliert werden können. Tiefgehende und weiterführende Informationen zum EPC können in dem durch Intel veröffentlichten *Software Developer's Manual*[55] oder in der Arbeit von *Costan und Devadas*[56] gefunden werden.

Daten einer SGX-Enklave können in einem der drei folgenden Speicherarten abgelegt sein:[57]

- *Innerhalb der CPU* selbst sind die Daten stets im Klartext vorhanden. Die Daten sind in den CPU-Registern sowie in den Level-1 bis Level-3-Caches abgelegt. Dies geschieht solang der Prozessor im Enclave Mode ist und die dazugehörige Enklave ausgeführt wird.
- *Im Enclave Page Cache*, der spezielle Abschnitt im Arbeitsspeicher des Systems. Primär werden dort Programm- aber auch die Verarbeitungsdaten der SGX-Enklaven abgelegt. Diese Daten werden geschützt vor dem Zugriff durch externe Komponenten, wie Treibern oder dem Betriebssystem selbst. Der Speicher, welcher physisch zur Verfügung steht wird zwischen laufenden Enklaven verteilt. Jede Enklave kann nur auf den ihr zugewiesenen Speicher zugreifen.
- *In anderen RAM-Bereichen* können zusätzlich zum EPC Daten abgespeichert werden, sollten gewisse Enklaven einen erhöhten Speicherbedarf haben. Der Prozessor verwendet dazu den SGX-Treiber, er identifiziert selten genutzte Speicherseiten innerhalb des Enclave Page Cachses und lagert diese in den verbleibenden Hauptspeicher aus. Dieses Verhalten ähnelt dem Paging-Verfahren von RAM-Seiten in den

[55] Vgl. Intel Corporation (2022b).
[56] Vgl. Costan, V./Devadas, S. (2016).
[57] Vgl. Vaucher, S. et al. (2018), S. 731.

Swap-Speicher. Die zu schützenden Daten werden transparent durch die CPU verschlüsselt beim Auslagern oder wieder entschlüsselt beim Kopieren in den EPC, um die Vertraulichkeit zu wahren.

EPC Aufbau Der Enclave Page Cache bezeichnet einen gewissen Teil des Processor Reserved Memory (PRM) des Computersystems. Dieser Bereich ist wiederum ein eigener Bereich im Arbeitsspeicher, dieser wird von der verwendeten Firmware, bevor das eigentliche System startet, reserviert. Durch die Prozessortechnologie Memory Encryption Engine (MEE) wird dieser Speicherabschnitt in der gesamten Laufzeit verschlüsselt, dieser Zustand wird stets aufrecht gehalten. Für die CPU selbst ist diese Verschlüsselung transparent. Andere Software, auch mit weitergehenden Rechten kann nicht auf diesen Bereich zugreifen, nur die CPU selbst kann den PRM durch die MEE entschlüsseln und dementsprechend auslesen.[58]

Die Größe des Processor Reserved Memory wird im basic input/output system (BIOS) des Systems konfiguriert, weiterhin ist sie je nach verwendeter Hardware beschränkt. In der folgenden Tabelle wird der jeweilig zur Verfügung stehende EPC für verschiedene Prozessorgenerationen dargestellt:

Generation	6. Generation	10. Generation	3. Generation Xeon Scalable
Codename	Skylake	Icelake	Icelake
Veröffentlichungsjahr	2016	2019	2020
Anwendungsgebiet	Desktop/ Notebook	Desktop/ Notebook	Server/ Workstation
EPC-Größe	128 MB	256 MB	512 GB
Beispiel	i7-6700K	i7-1060G7	Xeon Platinum 8380

Tabelle 1: EPC-Größen im Verlauf der Intel Prozessor-Generationen (Quelle: eigene Darstellung)

In der sechsten bis zur neunten Generation sind von maximal verfügbaren 128 MB für den Enclave Page Cache nur bis zu 93,5 MB für die eigentlichen Daten innerhalb der Enklave nutzbar. Der restliche Speicher wird für Metadaten gebraucht.[59] Intel verdoppelte die verfügbare Größe des PRM in der zehnten Generation ihrer Prozessoren. Die dritte Generation der Xeon Scalable Prozessoren trägt ebenso den Codenamen Icelake. Bei diesen Workstation CPUs beträgt die maximale EPC-Größe 512 GB. Wird ein Dual-

[58] Vgl. Minkin, M. (2018), S. 8.
[59] Vgl. Vaucher, S. et al. (2018), S. 731.

Socket-Mainboard, das bedeutet zwei CPUs in einem System, verwendet, ist die maximal nutzbare Größe des EPC 1 TB.

Der Enclave Page Cache wird in Speicherseiten unterteilt, jeweils mit einer Größe von 4 KB, diese werden dynamisch belegt. Jede Speicherseite wird genau einer Enklave zugeordnet, diese wird als Owning Enclave (OE) bezeichnet. Damit diese EPC-Seiten verwaltet werden können, existiert eine Struktur im Processor Reserved Memory, die sogenannte Enclave Page Cache Map (EPCM), außerdem können dadurch Zuordnungen im Enklavenspeicher überprüft werden. Pro Speicherseite des Enclave Page Cache enthält sie einen Eintrag. In diesem ist unter anderem der Status der Zuordnung gespeichert, ihr Typ und die OE wird referenziert.[60] Der Aufbau des Enclave Page Caches sowie den damit verbunden Speicherbereichen wird in der folgenden Abbildung 7 zusammengefasst:

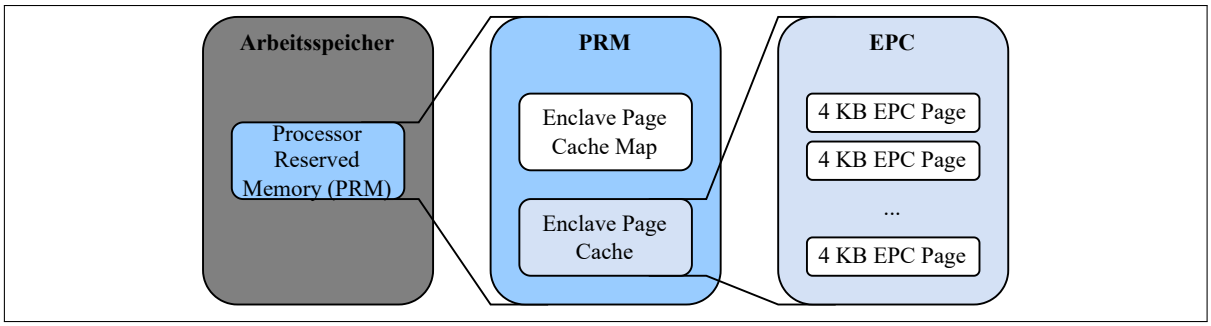


Abbildung 7: Aufbau des Enclave Page Cache (Quelle: eigene Darstellung)

Sicherheitsmechanismen Durch das Verwenden vom EPC, als dedizierten Speicher der Enklave werden die Schutzziele Integrität und Vertraulichkeit erreicht. Dabei soll nur eine Enklave die Möglichkeit besitzen auf ihre Daten zuzugreifen, welche im Arbeitsspeicher gespeichert sind. Durch die Software Guard Extensions werden die Daten der Enklave stets nur verschlüsselt in den Hauptspeicher gelegt. Mittels der Memory Encryption Engine des Prozessors werden diese Daten vor der Verwendung entschlüsselt. Die CPU hat dabei die Hoheit über die Zugriffe und kann diese verwalten und wahrt die Exklusivität der Nutzung durch die spezielle Enklave.

Dem Betriebssystem wird grundlegend misstraut, so auch dem Hypervisor, dieses ist für die Speicherverwaltung verantwortlich, einschließlich De-/Allokation sowie das Paging von Seiten, im Enclave Page Cache. Dadurch das alle Anweisungen durch die CPU ausgeführt werden und besondere SGX-Instruktionen benötigen, kann dieser unter Verwendung der EPCM sicherstellen, dass bereits zugeordnete EPC-Seiten nicht einer anderen Enklave zugewiesen werden können, beispielsweise durch das Betriebssystem. Zunächst muss die Verbindung der Seite zur Owning Enclave aufgelöst werden, dadurch wird der Speicherinhalt durch den Prozessor gelöscht und überschrieben. Die Speicherung des Typs jeder

[60] Vgl. Costan, V./Devadas, S. (2016), 58 f. & 92.

Seite in der EPCM hat zur Folge, dass die Seite zu einem nicht vorhergesehenen Zweck missbraucht werden kann.

Das Betriebssystem ist außerdem dafür zuständig, dass der vertrauenswürdige Code in die zugeordneten Enclave Page Cache Seiten über ein Interface des Prozessors geladen wird. Zudem werden Schnittstellen und Dienste bereitgestellt, um die Enklave und zugehörigen Speicher zu verwalten.[61]

Ist es notwendig, dass eine Seite des EPC durch das Betriebssystem in den übrigen Arbeitsspeicher ausgelagert wird, müssen alle logischen Prozessoren, welche mit vertrauenswürdigen Code aus der zugeordneten Owning Enclave arbeiten, die betreffende Enklave verlassen. Dieses Verhalten ist in SGX vorgeschrieben und geschieht meist durch einen Interrupt, welcher wiederum zu einem Asynchronous Enclave Exit führt. Anschließend wird der dazugehörige Translation Lookaside Buffers (TLB) geleert, dadurch existieren keine Einträge mehr, welche auf die gelöschten EPC-Seiten verweisen könnten. Bevor eine Seite in ungeschützten Bereich des Speichers kopiert wird, wird dieser wieder durch die CPU verschlüsselt. Mit der Verwendung einer *Nonce*, der *page version* werden Replay-Angriffe verhindert.[62] Replay-Angriffe bezeichnen eine Gruppe von Angriffen in der kryptografischen Kommunikation. Der Angreifer greift eine Nachricht ab und versendet diese zu einem späteren Zeitpunkt, modifiziert oder unverändert, erneut oder blockiert sie komplett.[63]

Kryptografie Es werden viele verschiedene kryptografische Schlüssel und Verfahren in unterschiedlichen SGX-Funktionen eingesetzt. Viele Details zur Implementierung sind sehr komplex, daher werden in diesem Abschnitt nur diejenigen erläutert, welche für die wichtigsten Eigenschaften und Schlüssel erforderlich sind. Weitere Informationen können dem durch Intel veröffentlichten *Software Developer's Manual*[64] oder der Arbeit von *Costan und Devadas*[65] entnommen werden.

Schlüssel und Geheimnisse Die Intel Software Guard Extensions erreichen ihre Sicherheit größtenteils mittels kryptografischer Verschlüsselung. Während der Laufzeit einer Enklave generiert der Prozessor viele unterschiedliche Schlüssel, unter anderem den *ProvisioningKey*, *LaunchKey* oder *AttestationKey*. In diesem Abschnitt werden nur die grundlegenden Prinzipien vorgestellt, auf diesen basieren dann alle weiteren Schlüssel.

---

61 Vgl. Costan, V./Devadas, S. (2016), S. 58 f.
62 Vgl. ebenda, S. 69 ff.
63 Vgl. Luntovskyy, A./Gütter, D. (2022), S. 287.
64 Vgl. Intel Corporation (2022b).
65 Vgl. Costan, V./Devadas, S. (2016).

Im Prozess der Herstellung eines Prozessors, welcher SGX-fähig sein soll, wird dieser mit zwei Hardwareschlüsseln ausgestattet, diese befinden sich in bestimmten Speichermodulen, welche unveränderlich sind und nur von der CPU selbst gelesen werden können. In der Arbeit von *Costan und Devadas*[66] wird vorgeschlagen, diese Schlüssel als *Secrets* zu bezeichnen, da diese in den Verfahren zur Verschlüsselung und Signatur niemals ohne Veränderung eingesetzt werden, vielmehr werden sie als *Wurzelschlüssel* (engl.: *Root Keys*) eingesetzt, um weitere Schlüssel abzuleiten.

Das *ProvisioningSecret* wird in einer eigenen Intel Key Generation Facility (iKGF), während der Herstellung der CPU generiert, zudem werden sie durch Intel in eine Datenbank abgespeichert. Der Hersteller und die physische CPU teilen dieses Geheimnis und nutzen es beispielsweise zur Authentifizierung der SGX-Plattform gegenüber Intel. Der Nutzer kann sich dadurch sicher sein, dass wirklich nur SGX-Prozessoren die Bestätigung als solche erhalten.[67]

Das *SealSecret* wird ebenfalls in dieser iKGF generiert, aber Intel versichert, dass jegliche Spuren und Hinweise des *SealSecret*s in dieser Fabrik gelöscht werden. Weiterhin versichern sie, dass es ausschließlich der CPU selbst bekannt sei. Die Schlüssel, welche nicht notwendig für die Kommunikation mit Intel sind, entstehen durch die Ableitung von diesem *SealSecret* Root Key.[68]

Der Hersteller Intel kann keinerlei Rückschlüsse auf diejenigen Schlüssel ziehen, welche, teilweise oder ganz, vom *SealSecret* abgeleitet werden, da dieser nur Kenntnis über das *ProvisioningSecret* hat, ein Beispiel wäre der *SealKey*.[69]

Datenstrukturen Für die Kryptografie der SGX-Funktionen werden neben den Schlüsseln weitere Dinge benötigt, wie dem Sealing oder Attestation. Es werden Metadaten und eigentliche Daten der kompletten TCB beziehungsweise der Enklave selbst eingesetzt, diese werden entweder in der CPU selbst oder im Enclave Page Cache gespeichert. Mittels dieser Daten kann eine Bindung mit den Eigenschaften der Enklave, welche auf Kryptografie beruht, erstellt werden. SGX-Prozesse verwenden komplexe Datenstrukturen zur Ein- und Ausgabe, diese kombinieren verschiedene solche Informationen.

Vier verschiedene Datenstrukturen und die dazugehörigen Daten sollen im folgenden kurz zum weiteren Verständnis gezeigt werden. Sie sind aus den Arbeiten von *Costan und Devadas*[70] und *Anati et. al.*[71] entnommen.

---

66 Vgl. Costan, V./Devadas, S. (2016), S. 85 f.
67 Vgl. Georgia Institute of Technology (2019c).
68 Vgl. Costan, V./Devadas, S. (2016), S. 85 f.
69 Vgl. ebenda.
70 Vgl. ebenda, S. 81 ff.
71 Vgl. Anati, I. et al. (2013), S. 2.

MRENCLAVE Identifikation einer Enklave mittels eines Hashs des vertrauenswürdigen Codes, den zur Erstellung vorhandenen vertrauenswürdigen Daten, der Position der Speicherseiten im Enclave Page Cache und zusätzlich der verwendeten Flags zur Initialisierung.

MRSIGNER Identifikation des Autors/ Erstellers einer Enklave mittels eines Hashs von seinem verwendeten öffentlichen Schlüssel.

REPORT Wird als Authentizitätsnachweis bei der lokalen Attestation verwendet, zwischen zwei Enklaven auf der selben Plattform.

QUOTE Wird als Authentizitätsnachweis bei der entfernten Attestation verwendet, zwischen einer Enklave und einer externen Komponente oder Nutzer, dieser kann nur signiert werden durch eine besondere Intel-Enklave.[72]

6.1.5 SGX-Umgebung

In diesem Abschnitt wird die Software-Seite beschrieben, welche vorhanden sein muss, neben eines unterstützten Prozessors, um Anwendungen in SGX-Enklaven auf einem Rechner auszuführen. Die Bezeichnung *SGX-Umgebung* umfasst dabei hier alle notwendigen Komponenten hinsichtlich der Software.

BIOS-Einstellungen Es müssen gewisse Einstellungen im BIOS, der Firmware des Systems, konfiguriert werden, um die Software Guard Extensions des Prozessors nutzen zu können. Im BIOS kann zunächst konfiguriert werden, ob SGX aktiviert, deaktiviert oder mittels einer Software aktiviert werden kann. In diesem letzten Fall haben Anwendungen zu Beginn keinen Zugriff auf SGX und den damit verbundenen Funktionen. Im nicht vertrauenswürdigen Teil einer Anwendung können bestimmte Funktionen aus Bibliotheken ausgeführt werden, um die SGX-Funktionen freizuschalten. Wird dieser Ansatz verwendet, muss die CPU den Speicherbereich des Processor Reserved Memory nicht immer bereithalten. Dieser wird vom gesamten Hauptspeichers des Systems abgezogen. Der Prozessor muss ihn dann nur bereitstellen, wenn es tatsächlich zur Ausführung einer SGX-Anwendung kommt.[73]

Im BIOS kann die maximale Größe des PRM angepasst werden und damit entsprechend auch die maximale Größe des Enclave Page Caches.

[72] Vgl. Georgia Institute of Technology (2019c).
[73] Vgl. Mechalas, J. P. (2017).

SGX-Treiber Der Zugriff auf Hardwarefunktionen von SGX erfordert einen speziellen eigenständigen Treiber. Die eigentliche Plattformsoftware gibt Anweisungen an den Treiber weiter, um mit dem Prozessor zu interagieren, dadurch können beispielsweise Enklaven erstellt werden oder die CPU wechselt in den Enclave Mode. In Windows kann SGX mittels eines Treiberpaketes installiert und verwendet werden. Für Linux wird ein Kernelmodul[74] angeboten, welches nachgeladen werden kann. Beginnend mit der Linux Kernel Version 5.11[75] ist der Treiber bereits im Kernel vorhanden und eine zusätzliche Installation ist nicht notwendig.

Plattformsoftware In der Intel SGX Plattformsoftware (PSW) werden verschiedene Komponenten der Software zusammengefasst. Sie stellt die für Enklaven notwendige Laufzeitumgebung zur Verfügung. Diese Eigenschaften machen sie absolut notwendig für die Ausführung einer Anwendung mittels SGX. Es sind Bibliotheken enthalten, welche für das Starten und die Verwaltung von Enklaven zuständig sind, zusätzlich können sie mit dem SGX-Treiber kommunizieren. Außerdem sind in der PSW sogenannte *Architectural Enclaves (AEs)* enthalten, sowie der benötigte Systemdienst Architectural Enclave Service Manager (AESM), um auf die notwendigen AEs zu zugreifen seitens der SGX-Anwendung. Der Hersteller Intel hat viele AEs entwickelt und stellt diese signiert bereit, um gewisse Aufgaben in der Verwaltung und Ausführung von Enklaven durchzuführen, welche genauso durch die Hardware-Umgebung geschützt werden.[76] Im Folgenden werden drei ausgewählte Architectural Enclaves kurz aufgezeigt:

Launch Enclave (LE) Diese Enklave startet vor allen anderen Enklaven und wird für den Start dieser benötigt. Dieser wird erst ermöglicht, wenn alle Checksummen und Signaturen überprüft und verifiziert wurden.

Quoting Enclave (QE) Bei der Entfernen Attestation wird diese Enklave genutzt, um den **REPORT** zu signieren und damit zu bestätigen, dass dieser lokal erstellt wurde.

Platform Service Enclaves (PSEs) Diese Enklaven fassen mehrere Architectural Enclaves zusammen, welche Funktionen für andere Enklaven bereitstellen, wie beispielsweise kryptografische oder mathematische Funktionen.

SGX SDK Ein Software Development Kit (SDK) ist die Zusammenfassung von Werkzeugen und Bibliotheken, welche zur Programmierung und Entwicklung von Software verwendet wird. Das SDK für SGX wird daher nur auf Rechnern gebraucht, welche zur Entwicklung genutzt werden. Es werden Bibliotheken bereitgestellt, welche zur Erstellung

[74] Vgl. Intel Corporation (2021a).
[75] Vgl. The kernel development community (2022).
[76] Vgl. Scarlata, V. et al. (2019), S. 3.

von Anwendungen, die SGX nutzen sollen, genutzt werden können, beispielsweise gewisse Kryptografiefunktionen. Im SGX SDK sind zudem Hilfsprogramme enthalten, die die Entwicklung unterstützen, zum Beispiel durch das Generieren von Proxy Functions oder dem Signieren von Enklaven. Zusätzlich können Enklaven damit debuggt werden. Ebenso sind Beispielcode und Vorlagen vorhanden, welche zur Orientierung und zum Einstieg genutzt werden können.

Intel veröffentlicht die Spezifikationen der API, der Programmierschnittstelle, hinsichtlich des Software Development Kits, für Windows[77] und Linux.[78]

6.1.6 SGX-Funktionen

Enklaven bieten viele sicherheitsrelevante Möglichkeiten und Funktionen, dabei sollen in diesem Abschnitt die wichtigsten beschrieben und erläutert werden. Um das Vertrauen in die Technologie hinsichtlich der Sicherheit zu haben, sind diese eine Voraussetzung dafür.

Enclave Attestation Um dem Problem des sicheren Rechnens auf entfernten Rechnern durch die vertraute Hardware zu begegnen, wird das Vertrauen in die Art und Weise der Funktion der genutzten Soft- und Hardwarekomponenten gebraucht und demzufolge auch dem Hersteller gegenüber. Bei SGX sind diese Komponenten die CPU, samt Memory Encryption Engine, der verwendete Treiber und alle Softwarebibliotheken, zum Beispiel einschließlich genutzter AEs. Damit ein Nutzer bereits vor dem Betreten einer abgesicherten Enklave eine Manipulation oder Veränderung ausschließen kann, muss er den vertrauenswürdigen Code innerhalb einer Enklave hinsichtlich seiner Integrität überprüfen können. Um die Grundlagen, auf welchen das Vertrauen beruht, zu beweisen, wird die *(Software) Attestation* bei Intel SGX verwendet. Dabei wird zum Attestieren auf der selben Platform, der gleiche Prozessor, einer Enklave im Kontakt zu einer zweiten Enklave die *Local Attestation* verwendet. Nutzt die Partei, welche eine Überprüfung durchführen möchte einen externen Rechner, kommt die *Remote Attestation* zum Einsatz.[79]

Wurde die Integrität durch eines dieser zwei Verfahren verifiziert, können erst die vertrauliche Daten, wie beispielsweise kryptografische Schlüssel, der betreffenden Enklave zur Verfügung gestellt werden. Man nennt diesen Vorgang *Secret Provisioning.*

Local Attestation Bei der lokalen Attestierung kann eine Enklave die Authentizität einer weiteren aktiven Enklave überprüfen. Voraussetzung ist das beide Enklaven auf der

[77] Vgl. Intel Corporation (2020c).
[78] Vgl. Intel Corporation (2020a).
[79] Vgl. Anati, I. et al. (2013), S. 2.

gleichen physischen CPU ausgeführt werden. Bei der Prüfung werden folgende Informationen verwendet, entnommen aus den Arbeiten von *Costan und Devadas*[80] und *Anati et. al.*[81]:

- Die Identität der SGX-Enklave (**MRENCLAVE**)
- Die Identität des Autors/ Erstellers der SGX-Enklave (**MRSIGNER**)
- Die Metadaten der SGX-Enklave
- Die Daten des Nutzers
- Die Identität der verwendeten SGX-Plattform, einschließlich des *SealSecret* und *ProvisioningSecret*

Diese Daten werden in der Datenstruktur **REPORT** zusammengefasst. Sie wird ganz speziell für eine verifizierende und eine attestierende Enklave erstellt, sodass nur die verifizierende Enklave den **REPORT** überprüfen kann. Dieser kann nur bestätigt werden, wenn die gleiche CPU für beiden Enklaven verwendet wird, da die SGX-Plattform mit ihrer Identität in den Generierung einfließt. Optional können die Daten des Nutzers verwendet werden, um einen Austausch von Schlüsseln zwischen den beiden Enklaven zu ermöglichen. Nach erfolgter Attestierung können die ausgetauschten Schlüssel weiter genutzt werden, um beispielsweise die Kommunikation abzusichern.[82] Zum Beispiel kann das Diffie-Hellman-Protokoll[83] zum Schlüsselaustausch verwendet werden.

Remote Attestation Kommt es zur Attestierung einer Enklave auf einer entfernten Plattform, beispielsweise in einer öffentlichen Cloud, können keine gemeinsamen Hardware-Geheimnisse als Grundlage für das Vertrauen verwendet werden. An diesem Punkt wird eine externe Stelle, welche zur Vermittlung genutzt wird, gebraucht. Mit Hilfe des abgelegten *ProvisioningSecrets* kann Intel, als Hersteller, die Integrität der Plattform validieren. Bei der Attestierung auf externen Rechnern wird Intel damit zur Vermittlungsstelle. Bei der Remote Attestation wird das **QUOTE** erstellt und normalerweise über das Internet mit dem Intel Attestation Service (IAS) verifiziert. Werden, wie beispielsweise in Rechenzentren, Systeme ohne Internetzugang, als Offline-Rechner bezeichnet, verwendet, können alternativ auch eigene Verifikationsserver eingesetzt werden, diese nutzen die sogenannten Data Center Attestation Primitives (DCAP).[84]

[80] Vgl. Costan, V./Devadas, S. (2016), S. 84.
[81] Vgl. Anati, I. et al. (2013), S. 2 f.
[82] Vgl. Georgia Institute of Technology (2019c).
[83] Vgl. Diffie, W./Hellman, M. E. (1976).
[84] Vgl. Scarlata, V. et al. (2019).

Intel Attestation Service Um die Authentizität einer Enklave auf einem externen Rechner zu prüfen, wird die im Abschnitt 6.1.5 genannte **Quoting Enclave** genutzt. Sie arbeitet zwischen der Enklave und derjenigen Partei, welche überprüfen möchte und attestiert die Enklave innerhalb des lokalen Systems, danach generiert sie die **QUOTE** und sendet diese an die überprüfende Partei.

Außerdem wird eine Attestierung, durch Intel, der QE benötigt, damit die gesamte SGX-Umgebung als authentisch verifiziert werden kann. Dieser Prozess, unterstützt durch die Architectural Enclaves, wird als *Provisioning* bezeichnet. Es wird das Protokoll Enhanced Privacy ID (EPID)[85] verwendet, was mit Fokus auf den Datenschutz optimiert ist.

Intel Data Center Attestation Primitives Unter Umständen ist nicht die Möglichkeit gegeben, eine Internetverbindung zum IAS für die Remote Attestation zu nutzen oder nicht gewünscht. Intel führte 2018 alternativ die Data Center Attestation Primitives ein, dadurch können eigene Verifikationsserver zur Offline Attestierung eingesetzt werden. Die Funktion ist ausschließlich bei Workstation-Prozessoren vorhanden, den sogenannten Intel Xeon-Reihen.[86] Bevor DCAP produktiv eingesetzt und verwendet werden kann, ist eine initiale Kommunikation mit den Servern von Intel notwendig, dadurch wird weiterhin ein gewisses Vertrauen in Intel als Hersteller vorausgesetzt und man ist nicht vollständig unabhängig von diesem.[87]

Bei der vorhergehenden Methode der Remote Attestation kommt der IAS zum Einsatz, um die **QUOTE** zu verifizieren, in diesem Szenario wird der eigene Verifikationsserver dafür verwendet. Dieser Server braucht eine eigene Datenbank mit Zertifikaten aller CPUs der verwendeten Plattformen, um einen Abgleich durchzuführen. Diese Datenbank muss von Intel verifiziert werden, anschließend wird sie signiert.[88] Die Trennung vom Internet kann daraufhin erfolgen und die Authentizität von Enklaven kann mittels dieser Technologie überprüft werden.

Um dieses Verfahren zum Attestieren zu Verwenden sind diese drei folgenden Voraussetzungen zu erfüllen:

- Nutzung einer CPU, welche DCAP fähig ist, außerdem muss die Funktion *Flexible Launch Control* aktiv sein.[89]
- Der SGX-Treiber muss eine Version mit DCAP-Unterstützung sein.
- Verwendung einer speziellen **DCAP-Quoting Enclave**[90]

[85] Vgl. Brickell, E./Li, J. (2007).
[86] Vgl. Johnson, S. P. (2018).
[87] Vgl. Scarlata, V. et al. (2019), S. 3 ff.
[88] Vgl. Intel Corporation (2022a).
[89] Vgl. Johnson, S. P. (2018).
[90] Vgl. Scarlata, V. et al. (2019), S. 4 f.

Sealing Werden Daten einer Enklave auch nach dem Ende von ihr benötigt, so ist es notwendig, dass diese kryptografisch „versiegelt“ werden, um sie persistent zu haben, bevor die eigentliche Enklave beendet wird. Intel bezeichnet diese Speicherung der Daten für den Austausch zwischen Enklaven *Sealing*.

Die vertrauenswürdigen Daten werden während dieses Vorganges in der Enklave selbst verschlüsselt und mit der Hilfe des nicht vertrauenswürdigen Teils der Anwendung persistent auf der Festplatte abgelegt. Sollen die Daten innerhalb einer anderen Enklave verwendet werden, erfolgt nach dem Auslesen vom Massenspeicher die Entschlüsselung mit dem gleichen Schlüssel, das sogenannte *Unsealing*. Die Enklave kann selbst das Verschlüsselungsverfahren wählen. Um Replay-Angriffe zu verhindern, empfiehlt Intel beispielsweise zusätzlich die Verwendung von *Nonces*.[91]

Beim Sealing wird der *SealKey* verwendet zum Ver- und Entschlüsseln, er wird von beiden existierenden Hardware-Secrets abgeleitet, dadurch kann Intel, als Hersteller diesen nicht generieren. Der Entwickler kann die Erstellung des SealKeys beeinflussen, indem er eine *Key Policy* wählt, daraus folgen auch Eigenschaften für das Sealings.

Sollen Daten sicher nur von einer Enklave, mit mehreren Instanzen, ausgetauscht werden, muss der *SealKey* mittels Kryptografie an die Identität, **MRENCLAVE**, der Enklave gebunden werden. Wird der Austausch vertrauenswürdiger Daten über mehrere Enklaven gewünscht, kann die Identität des Autors, **MRSIGNER**, für die Ableitung des Schlüssels genutzt werden.

Zusätzlich zur *Key Policy* wird die aktuelle Enklaven-Versionsnummer mit im *SealKey* verarbeitet. Dies hat zur Folge, dass wenn eine SGX-Anwendung aufgrund eines Sicherheitsvorfalles geupdatet wurde, eine veraltete aber noch existierende Enklave nicht auf die neuen vertrauenswürdigen Daten Zugriff hat. Der Prozessor gewährleistet hingegen, dass neue Enklaven auf ältere Daten zugreifen können.[92]

6.1.7 Erweiterungen und Einschränkungen

Die gesteigerte Sicherheit durch Intel SGX geht mit gewissen Einschränkungen hinsichtlich der Nutzungsfreundlichkeit und Ausführungsgeschwindigkeit einher. Gewisse Nachteile in der ersten Generation von Intel SGX wurden bereits verbessert oder ganz behoben.

Langsames EPC-Paging Wenn eine Enklave einen höheren Speicherbedarf hat, als ihr im Enclave Page Cache zur Verfügung steht, werden wenig genutzt Speicherseiten

[91] Vgl. Anati, I. et al. (2013), S. 4.
[92] Vgl. ebenda, S. 4 f.

in den nicht abgesicherten Arbeitsspeicher ausgelagert. Damit die gewonnene Sicherheit seitens Intel SGX gewahrt wird, muss vor dem Paging eine Verschlüsselung und nach dem Zurückkopieren eine Entschlüsselung erfolgen. Außerdem gibt es Maßnahmen zur Sicherung der Integrität durch den Prozessor und der verwendeten Memory Encryption Engine, diese verringern die Ausführungsgeschwindigkeit der Enklave.

Dieses Problem trat vor allem bei Systemen der sechsten bis zur zehnten Generation von Intels Prozessoren auf, da diese nur 128 MB respektive 256 MB EPC-Kapazität hatten. Der beste Weg um diesen Problem zu entgehen war und ist die Reduzierung der Trusted Computing Base, also die Größe der gesamten Enklave.

Fehlende Systemcalls Dem Kernel des Systems wird bei SGX misstraut, er ist damit außerhalb der TCB. Kernelfunktionen werden für Systemcalls gebraucht, die Rückgabewerte von ihnen können daraufhin verändert werden, deshalb sind jene Aufrufe von Funktionen in der Enklave untersagt.[93] Der Zugriff auf Ein- und Ausgabe-Ressourcen ist damit nicht möglich.[94]

Wird die Funktion eines solchen Systemcalls in einer Enklave gebraucht, so muss diese einen Outside Call in den nicht vertrauenswürdigen Code und dieser dann den Systemcall selbst ausführen. Um die Rückgabewerte danach wieder weiterzuleiten muss ein Enclave Call durchgeführt werden. Dieses beschriebene Verhalten führt zu einer deutlich sinkenden Geschwindigkeit des vertrauenswürdigen Codes, da es stets Sicherheitsmaßnahmen beim Verlassen und Betreten der Enklave gibt.

Es wurden unterschiedliche Ansätze erforscht, um performantere Systemcalls zu erreichen[95]. Im Jahr 2018 hat Intel die *Switchless Calls* im offiziellen Software Development Kit von SGX eingeführt, alternativ zum oben beschriebenen Vorgehen, diese nutzen Remote Procedure Call (RPC), um entfernte Prozeduren aufzurufen.[96]

Dabei übernimmt weiterhin der nicht vertrauenswürdige Programmcode der Anwendung den Aufruf von Kernel-Code, der Vorteil ist, dass nicht mehr aus der Enklave gewechselt werden muss. Eingesetzt werden Anwendungsthreads, welche asynchron laufen, diese führen den Systemcall durch, mittels eines Speicherbereichs, der zwischen vertrauenswürdigen und nicht vertrauenswürdigen Programmkomponente geteilt ist, findet die Kommunikation und Übergabe von Parametern der Funktion und deren Rückgaben statt. Die eingesparten Schritte ermöglichen eine schnellere Ausführung der Enklave.[97]

---

93 Vgl. Intel Corporation (2020a), S. 35.
94 Vgl. Costan, V./Devadas, S. (2016), S. 90.
95 Vgl. Arnautov, S. et al. (2016).
96 Vgl. Luntovskyy, A./Gütter, D. (2020), S. 340.
97 Vgl. Tian, H. et al. (2018), S. 22 ff.

Statische Speicherzuordnung Die erste Version von SGX erforderte, dass der komplette Speicherbedarf der Enklave zugeordnet wird, bevor sie überhaupt initialisiert wurde, dafür mussten alle Berechtigungen konfiguriert werden. Dies war erforderlich, damit alle zugeordneten Seiten im Enclave Page Cache und die dazugehörige Konfiguration im **MRENCLAVE**-Hash verarbeitet werden konnten. Das Laden im Nachhinein von Code oder vertrauenswürdigen Daten noch das Lösen von nicht mehr gebrauchten EPC-Speicher, welcher durch andere Enklaven benutzt werden könnte, war dadurch nicht möglich. 2016 wurde dann die zweite Version der Software Guard Extensions veröffentlicht, welche weitere Instruktionen für den Prozessor enthält, um den Speicher dynamisch und sicher zu verwalten.[98]

Remote Attestation und Internetzugriff Die Kommunikation für die Remote Attestation erforderte zu Anfang eine Verbindung zum Internet. Im Abschnitt 6.1.6 wurde erläutert, dass Intel es erlaubt eigene Verifikationsserver mit den DCAP einzusetzen, um die Attestierung innerhalb eines privaten und getrennten Netzwerks zu ermöglichen.

Die Software Guard Extensions bieten keine Möglichkeit die Vertraulichkeit des Programmcodes innerhalb der Enklave zu überprüfen. Mit Hilfe des Hashs kann die Integrität dessen sichergestellt werden, beim Laden aus der nicht vertrauenswürdigen Komponente der Anwendung in die Enklave. Ein Angreifer kann aber Zugriff auf den Inhalt erlangen. Das Kritische an diesem Problem kann durch die Verwendung von einem Hilfsprogramm verringert werden. So wird bei der Erstellung der Enklave dieses in sie geladen, mit Überprüfung der Integrität. Dieses Programm lädt nach dem Start der Enklave den vertrauenswürdigen Code nach und gibt Enclave Call Funktionsaufrufe weiter. Dadurch kann der Programmcode der Enklave auf einem vertrauten System zur Verfügung gestellt werden, dort kann dieser verschlüsselt abgelegt sein. Damit wird die Vertraulichkeit bei der Übertragung gewahrt. Das Hilfsprogramm übernimmt die Integritätsprüfung des Codes.

6.1.8 Migration von Anwendungen

Wird eine Anwendung zur Nutzung von Intel SGX neu entwickelt, können die Besonderheiten der Enklaven bereits von Beginn an berücksichtigt werden, zum Beispiel die Unterteilung in vertrauenswürdigen und nicht vertrauenswürdigen Programmcode und das Konzept zur Kommunikation mit der Enklave, wie in Abschnitt 6.1.3 beschrieben.

Um bestehende Software, auch nur ein Teil des Codes, in einer SGX-Enklave auszuführen, ist eine Anpassung dessen notwendig, um die speziellen Eigenschaften der Software Guard Extensions zu erfüllen. Diese kann die Anwendung nicht selbst bereitstellen, wenn es zu

[98] Vgl. McKeen, F. et al. (2016), S. 1 f.

möglich geringen Anpassungen kommen soll. Vielmehr werden, teilweise automatisiert, Veränderungen und Erweiterungen des Codes benötigt. Durch externe Werkzeuge und Dienste können die gebrauchten Funktionen zur Verwaltung oder die Kommunikationsschnittstellen der Enklaven zur Verfügung gestellt werden.

Damit Entwickler diese Herausforderungen leichter lösen können, sind zahlreiche wissenschaftliche Arbeiten und Studien hinsichtlich verschiedener Herangehensweisen veröffentlicht wurden. Zudem haben mehrere private Unternehmen Produkte und Dienstleistungen zur Migration von herkömmlichen Anwendungen entwickelt und bieten diese an. In diesem Abschnitt wird ein Überblick über die Ansätze gegeben.

Strategien Die verschiedenen Ansätze der Migration verfolgen unterschiedliche Ziele und setzen andere Prioritäten. Beispielsweise versuchen einige die Aufteilung einer bestehenden Anwendung in vertrauenswürdigen und nicht vertrauenswürdigen Teil zu vereinfachen, wiederum versuchen andere die gesamte Anwendung in einer SGX-Enklave auszuführen.

Im Folgendem werden die verschiedenen Ansätze der Migration kategorisiert, je nach Strategie. Mit sinkendem Aufwand manueller Anpassungen geschieht das wie folgt.

Programmierbibliothek Um SGX-Anwendungen zu programmieren, kann eine Bibliothek verwendet werden, dabei wird der bereits existierende Code erweitert, um die benötigten Funktionen für eine Enklave zu modifizieren beziehungsweise zu erweitern.

Durch die Verwendung dieses Ansatzes kommt es meistens zum größten Aufwand einer Migration. Sie eignet sich in der Regel am wenigsten, vor allem wenn sehr große und komplexe Anwendungen migriert werden sollen. Wird dieser Ansatz aber nicht allein eingesetzt, sondern begleitend zu einer Alternative, kann die Sicherheit, Geschwindigkeit und auch die Funktion der Enklave erheblich gesteigert werden.[99]

Automatisierungsframework Ein Framework kann verschiedene Aufgaben automatisieren und zur Migration einer Anwendung verwendet werden. Meist wird *Boilerplate Code* generiert, sodass dieser nicht mehrmals neu generiert werden müsste. Als *Boilerplate Code* werden die Teiles des Programmcodes genannt, welche vier Eigenschaften haben:[100]

1. Geringer Einfluss auf die Funktionalität
2. Häufig verwendet.

[99] Vgl. Orenbach, M. et al. (2017).
[100] Vgl. Nam, D. (2019), S. 1253.

3. Lokalität.

4. Sehr geringe Unterscheidungen zwischen unterschiedlichen Vorkommen davon.

Beispielsweise können Anwendungen, zum Teil oder vollständig, automatisiert in vertrauenswürdigen und nicht vertrauenswürdigen Code geteilt werden. Außerdem ist es möglich zusätzliche Programmteile zu generieren, welche Funktionen, wie Systemcalls oder die Überprüfung von Rückgabewerten übernehmen.[101]

Shielding Layer Der *Shielding Layer*, zu deutsch Abschirmschicht, kann verwendet werden, um eine gesamte Anwendung innerhalb einer Enklave auszuführen. Dabei agiert er als Vermittler zwischen des Programmcodes in der Enklave und den Schnittstellen des Betriebssystems außerhalb von ihr. Alle Aufrufe von Funktionen, welche von der Enklave selbst nach außen, wie beispielsweise Systemcalls, erfolgen, werden abgefangen und an den Gegenspieler im nicht vertrauenswürdigen Code weitergeleitet und verarbeitet. Dadurch wird der vertrauenswürdige Code abgeschirmt. Dies kann durch die Verschlüsselung von Paketen geschehen, welche transparent erfolgt oder Rückgabewerte können hinsichtlich ihrer Integrität überprüft werden.

Die Größe der Trusted Computing Base wird nur etwas vergrößert, dafür entsteht aber eine größere Schnittstelle zum unvertrauten Teil des Systems selbst.[102]

Library OS Um bestehende Anwendungen mit möglichst geringen Aufwand, ohne Veränderung und als Gesamtheit in einer Enklave auszuführen, können Betriebssysteme eingesetzt werden, welche auf Bibliotheken basieren. So ein *Library OS* enthält unterschiedliche Softwarebibliotheken, diese stellen Funktionen eines herkömmlichen Betriebssystems nach, anschließend sind sie innerhalb der Enklave verfügbar und können genutzt werden.

Ähnlich zum Shielding Layer kommt eine Schicht innerhalb der Enklave zum Einsatz, um Funktionsaufrufe von externen Funktionen in der Enklave abzufangen und je nach Bedarf an den nicht vertrauenswürdigen Teil der Anwendung weiterzuleiten. Ein Library OS enthält, hingegen zum vorhergehenden Ansatz, eine deutlich größere Funktionalität, mit dem Ziel einen Großteil der Funktionsaufrufe eigenständig abzuarbeiten, damit die Enklave nur verlassen werden muss, wenn es unbedingt notwendig ist. Dies geschieht, da Ein- und Ausgaben sehr aufwendig sind, wie im Abschnitt 6.1.7 beschrieben.

Die Größe der TCB steigt sehr stark, dafür wird die Schnittstelle zum nicht vertrauenswürdigen Programmteil und dem Betriebssystem verkleinert.[103]

[101] Vgl. Weisse, O./Bertacco, V./Austin, T. (2017).
[102] Vgl. Arnautov, S. et al. (2016), S. 692.
[103] Vgl. Tsai, C.-c./Porter, D. E./Vij, M. (2017), S. 645 ff.

Zusammenfassend lässt sich die Portierung einer bisherigen Anwendung durch die Verwendung eines Shielding Layers oder eines Library OSs am einfachsten umsetzen, je nachdem ob mehr Fokus auf die Größe der Trusted Computing Base oder der Schnittstelle zum Betriebssystem gelegt wird. Programmierbibliotheken sowie Frameworks erfordern mehr Aufwand hinsichtlich der Migration, auch wenn teilweise das Vorgehen automatisiert abläuft. Die Wahl des verwendeten Ansatzes ist abhängig vom speziellen Anwendungsfall selbst. Es können auch mehrere Strategien kombiniert werden, dadurch ergeben sich vielseitige Möglichkeiten.

Vorhandene Ansätze Im Folgenden werden verschiedene Kriterien erläutert, diese werden zum Vergleich unterschiedlicher Ansätze genutzt. Ausgewählte sind nach ihren Strategien in den Tabellen eins bis vier im Anhang geordnet und verglichen. Konnten Informationen nicht durch die Verwendung der vorhandenen Studien sowie weitergehender Recherchen ermittelt werden, wird dies mit einem Fragezeichen in den Tabellen markiert.

TCB Aus Gründen der Sicherheit und Geschwindigkeit sollte der Bedarf an Speicher einer Enklave so klein wie möglich sein. Darum ist es notwendig, dass die Größe der Trusted Computing Base der migrierten Anwendung möglichst wenig wächst.

Um einschätzen zu können, wie sich der Speicherbedarf verändert, wird die Zahl der Codezeilen angeführt, welche bei der Migration dem Programmcode hinzugefügt werden.

Open Source Ob der Code der verwendeten Strategie öffentlich einsehbar ist oder nicht, spielt je nach Verwendungszweck eine unterschiedliche große Rolle. Wird die Strategie produktiv eingesetzt, sollte auf eine länger gehende Unterstützung sowie kontinuierliche Weiterentwicklung und Pflege geachtet werden.

Aufwand Es wird eine qualitative Einschätzung hinsichtlich des Aufwandes der Migration gegeben. Die Einschätzungen reichen von *sehr gering* (dabei findet die Migration nahezu automatisch mit vereinzelten Eingriffen statt) bis *sehr hoch* (dabei sind viele Schritte extra notwendig und mit einem hohen zeitlichen und fachlichen Aufwand verbunden).

Systemcall-Ansatz Um die Sicherheit zu verbessern oder die Ausführung der Enklaven zu beschleunigen können die verschiedenen Migrationsansätze unterschiedliche Strategien verwenden, um Systemcalls aus der Enklave heraus auszuführen.

Systemcall-Unterstützung Nicht jeder Systemcall-Ansatz und die dabei genutzten Bibliotheken unterstützen alle Systemcalls, welche unter Linux verfügbar sind. Dadurch kann es zu Kompatibilitätsproblemen, vor allem bei älteren Anwendungen, kommen, sodass diese manuell modifiziert werden müssen, um sie zu beheben. Zusätzlich wird angegeben, welche Standard-C-Bibliothek für die Systemcall-Funktionen genutzt wird. Entweder die sehr umfangreiche und bekannte *GNU C Bibliothek*[104] oder die mit Fokus auf den Speicherbedarf verbesserte *musl libc*[105], welche veränderte Funktionalitäten inne hat.

Einschränkungen Werden Eigenschaften oder das Vorgehen von SGX durch die Migration verändert, kann es zu Einschränkungen während der Nutzung von Funktionen kommen. Manche Ansätze haben auch spezielle Anforderungen an die Anwendung oder an deren Umgebung.

Zusammenfassung Die in de Tabellen eins bis vier gezeigten Studien verfolgen jeweils verschiedene Ziele, sodass keine eindeutige Reihenfolge ihrer Wertigkeit zur Nutzung bei der Migration erstellt werden kann. Es muss stets für den Einsatzzweck spezifisch überprüft werden, welche Anforderungen seitens der Anwendung existieren und unter Nutzung welcher Ansätze diese so effizient wie möglich erfüllt werden können.

Ist die zu portierende Anwendung vergleichsweise klein und ist wenig komplex oder eine eigene Entwicklung, ist die manuelle Migration zu empfehlen mit dem Einsatz des Intel SGX SDKs[106] oder die Anpassung des Programmcodes durch die Verwendung von Eleos.[107] Dadurch lässt sich die Trusted Computing schmal halten und man erhält sehr gute Performanz.

Ist der damit verbundene Programmieraufwand zu groß und nicht möglich umsetzen, können Frameworks eingesetzt werden. Beispielsweise Glamdring[108] kann die Anwendung teilweise automatisch in den vertrauenswürdigen und nicht vertrauenswürdigen Teil aufteilen. Dafür ist keine Anpassung des Quellcodes selbst durch den Entwickler notwendig, dennoch muss er Kenntnis über diesen haben. Zum Beispiel müssen die Variablen erkannt werden, welche zu schützen sind und deswegen nur innerhalb der Enklave zur Verarbeitung bereitstehen dürfen.

Durch die Verwendung des Frameworks HotCalls[109] kann mittels der teilautomatisierten Migration die Performanz von durchgeführten Systemcalls gesteigert werden.

---

104 Vgl. Free Software Foundation, Inc. (2022).
105 Vgl. McClain, K. (2022).
106 Vgl. Intel Corporation (2020a).
107 Vgl. Orenbach, M. et al. (2017).
108 Vgl. Lind, J. et al. (2017).
109 Vgl. Weisse, O./Bertacco, V./Austin, T. (2017).

Mittels EGo[110] können bestehende Anwendungen, welche in der Programmiersprache Go geschrieben sind, migriert werden und so in einer SGX-Umgebung ausgeführt werden.

Ist die zu migrierende Anwendung sehr komplex und es fehlt an Kenntnissen über deren interne Spezifikationen kann eine Shieldiing Layer-Technologie eingesetzt werden, wie beispielsweise SCONE[111], um die Anwendung mit relativ überschaubaren Aufwand zu migrieren. Außerdem können weitere Sicherheitsfunktionen genutzt werden.

Mittels eines Library OSs kann bei der Migration die meiste Zeit eingespart werden. Die Anwendung wird recht ähnlich zu einem herkömmlichen Betriebssystem zur Ausführung gebracht. Die unterschiedlichen Ansätze verfolgen dabei verschiedene Ziele. Occlum[112] legt dabei seinen Fokus auf die Unterstützung von Multithreading und -tasking in der Enklave. Es ist zu beachten, dass Haven[113] als Library OS einzig und allein zum Ausführen von Windows-Anwendungen entwickelt wurde, jedes andere setzt ein Linux-System voraus.

Andere Lösungen Neben den gezeigten Ansätzen entwickeln auch verschiedene private Unternehmen Lösungen um Anwendungen in ein SGX-Umfeld zu migrieren. Kommerziell gesehen sind der Zeitaufwand und die damit verbundenen Kosten entscheidend, sodass viel Wert auf die schnelle und einfache Migration gesetzt wird. Die größten proprietären Ansätze werden innerhalb dieses Abschnittes vorgestellt.

Die Linux Foundation hat das Confidential Computing Consortium (CCC) erschaffen, welches sehr relevant in diesem Zusammenhang ist. Diese Vereinigung von Soft- und Hardware-Unternehmen sowie Dienstleistern versucht die Abstimmung und gemeinsame Weiterentwicklung zu fördern.[114] Im Folgendem werden einige Mitglieder und ihre Angebote präsentiert.

Public-Cloud-Anbieter Bestehende Cloud-Anbieter, wie Microsoft oder Google, vergrößern ihr Portfolio an Lösungen in der Cloud ständig. Es wird auch mehr Fokus auf Vertrauenswürdigkeit gelegt und damit auch auf die vertrauenswürdige Verarbeitung von Daten, durch die Verwendung von Technologien von Intel oder AMD. Microsoft bietet in der Azure-Cloud die Möglichkeit SGX-Prozessoren für virtuelle Maschine oder Kubernetes-Cluster zu verwenden.[115]

110 Vgl. Edgeless Systems GmbH (2022b).
111 Vgl. Arnautov, S. et al. (2016).
112 Vgl. Shen, Y. et al. (2020).
113 Vgl. Baumann, A./Peinado, M./Hunt, G. (2014).
114 Vgl. Confidential Computing Foundation (2022).
115 Vgl. Russinovich, M. (2018).

Google entwickelt eine Open-Source-Framework, namens Asylo, um bestehende Anwendungen zu portieren, dabei soll die spezielle Implementierung abstrahiert werden.[116]

Ant Group Das chinesische Unternehmen gehört zur Alibaba Group. Es entwickelt an einer Kombination von Containern mit Enklaven, die sogenannten *Inclavare Containers*. Dabei werden die Anwendungen in Container gekapselt und dann in dieser Laufzeitumgebung ausgeführt. Dafür werden bestehende Library OSs verwendet, wie zum Beispiel Graphene-SGX.[117]

Edgeless Systems Das deutsche Unternehmen stellt ein SDK, als Community Version, für die Öffentlichkeit zur Verfügung, dadurch können existierende Anwendungen einfacher migriert werden. Edgeless Systems stellt zudem weitere Softwarelösungen bereit, welche durch dieses SDK entwickelt wurden sind.[118]

Fortanix Diese Firma hat ein eigenes Library OS entwickelt, sodass bestehende Anwendungen ohne Änderungen des Codes migriert genutzt werden können, außerdem bieten sie verschiedene Dienstleistungen und Lösungen an. Beispielsweise haben sie eine Software entwickelt, mit welcher Enklaven zentral verwaltet und attestiert werden können.[119]

Das Confidential Computing Consortium umfasst neben kommerziellen Mitgliedern auch einige Open-Source-Projekte, wie zum Beispiel Graphene-SGX oder Occlum.

Betrachtungen zu der Anwendungspartitionierung Der Hersteller Intel entwickelt SGX um eine abgesicherte Umgebung für einen Teil einer Anwendung zu haben, in welchem vertrauenswürdige Daten verarbeitet werden. Dieser Teil soll möglichst klein und abgegrenzt sein. Es ist keine Stärke dieser Befehlssatzerweiterung gesamte Anwendungen innerhalb einer Enklave auszuführen, deshalb wurde der EPC zu Anfang mit relativ wenig Speicherplatz entwickelt.

Daraus folgend wird die höchste Performanz erreicht, wenn die existierende Anwendung, für die Verwendung von SGX gründlich manuell angepasst wird oder von Grund auf neu entwickelt.

Die Verkleinerung der Trusted Computing Base ist nicht nur ein Vorteil hinsichtlich der Geschwindigkeit der Anwendung, sondern trägt auch zur gesamten Sicherheit bei. Wird

[116] Vgl. Asylo Authors (2021).
[117] Vgl. The Inclavare Containers Authors (2021).
[118] Vgl. Edgeless Systems GmbH (2022a).
[119] Vgl. Fortanix (2022).

die Anwendung vor nicht berechtigten Zugriffen seitens des Betriebssystems geschützt, steht die Gesamtsicherheit in Zusammenhang mit dem vertrauenswürdigen Quellcode und den damit verbundenen Anweisungen. Die Wahrscheinlichkeit für eine Schwachstelle innerhalb dieses Codes steigt, je mehr Funktionen oder Bibliotheken innerhalb der Enklave sind.

Schwachstellen, welche bisher nicht bekannt sind, können in jeder Software vorkommen und können nicht grundsätzlich ausgeschlossen werden. Deshalb sollte nur diejenige Funktionalität in der Enklave ausgeführt werden, welche zwingend auf schützenswerte Daten zugreifen muss.

Fazit Eine strenge Aufteilung ist der beste Weg eine hohe Performanz und Sicherheit zu erreichen. Meist ist dieser Weg bei bestehenden Anwendung wenig praktikabel. Bei Open-Source-Anwendungen ist ein komplexes Studium der Struktur und des Codes notwendig, um zu erkennen, was geschützt werden muss. Bei Closed-Source-Anwendungen hingegen kann der Programmcode nicht selbstständig verändert werden.

6.1.9 Angriffe auf Intel SGX

In diesem Abschnitt werden zwei verschiedene Sicherheitslücken von Intel SGX kurz dargestellt. Aufbauend auf diesen Sicherheitslücken haben Forscher Angriffe entwickelt. Intel als Hersteller wurde vor der Veröffentlichung informiert und entwickelte Gegenmaßnahmen, wie beispielsweise Patches.

Foreshadow Anfang 2018 entdeckten zwei verschiedene Forscherteams unabhängig voneinander diese Sicherheitslücke. Normalerweise ist beim Versuch Enklavenspeicher auszulesen spekulative oder vorhersehende Ausführung möglich um den Cache zu verändern, je nachdem welche Daten gelesen wurden sind. Der Prozessor erkennt, dass Enklavenspeicher involviert ist und blockiert diese spekulative Ausführung, dadurch ist kein Auslesen möglich. Befinden sich diese sensitiven Daten aber im Layer 1 Cache der CPU, kann die Ausführung stattfinden, bevor der Prozessor feststellt, dass die Berechtigungen für diesen Vorgang fehlen. Dies ist der Foreshadow Angriff.[120]

Intel bezeichnet diesen Fehler als *L1 Terminal Fault* und veröffentlichte im August 2018 ein entsprechendes Update des Microcodes.[121]

[120] Vgl. Bulck, J. V. et al. (2018).
[121] Vgl. Intel Corporation (2021c).

Plundervolt Im Juni 2019 berichtete eine Gruppe internationaler Forscher eine Sicherheitslücke in den Software Guard Extensions an Intel als Hersteller, sie wird als Plundervolt bezeichnet. Prozessoren steuern ihre Frequenzen und Spannungen selbstständig. Bestimmte Spannungen können durch den Nutzer vorgegeben werden, andere wiederum nicht. Das Team entdeckte im Zuge von Reverse Engineering versteckte Register, welche bestimmte Spannungen steuern. Diese sind zugänglich, wenn man Zugriff auf das Betriebssystem hat. Der Angriff funktioniert so, dass mittels dieser versteckten Register die Spannung an der CPU genau im richtigen Moment ein wenig verändert wird, sodass die Enklave bei der Ausführung einen Fehler macht. Das Ziel des Angriffes ist die Manipulation von mathematischen Operationen, konkreter der Multiplikation. Durch das minimale Anheben oder Senken der Spannung kam es zu kleinsten Veränderungen auf Bit-Ebene. Nach eingehender Recherche waren Muster der Fehler erkennbar und man konnte Vorhersagen über das Fehlverhalten treffen, dadurch konnten kryptografische Schlüssel ausgelesen werden. Zudem wurde die Integrität der geschützten Speicherbereiche angegriffen, da Werte in diesem verändert wurden. Es war auch möglich den Angriff aus der Ferne durchzuführen.[122]

Im Dezember des gleichen Jahres veröffentlichte Intel ein Update für den Microcode der Prozessoren, welches dieses dieses Problem behebt.[123]

6.2 Weitere Technologien

6.2.1 AMD SEV

Im Jahr 2016 stellte AMD die Secure Memory Encryption (SME) und die Secure Encrypted Virtualization (SEV) vor. Beide Technologien bauen auf zwei Hardware-Lösungen auf. Einmal der AES-128 Hardware Verschlüsselungsengine und andererseits dem AMD Secure Processor, kurz AMD-SP. Erstere befindet sich im Speichercontroller, dieser verschlüsselt Daten, welche in den Hauptspeicher geschrieben werden und beim lesen entschlüsselt er diese wieder. Da der Speichercontroller sich direkt im System-on-a-Chip (SoC) des Prozessors befindet, enthalten alle Datenverbindungen welche die CPU verlassen nur verschlüsselte Daten. Der AMD Secure Processor ist eine dedizierte Komponente, welche kryptografische Funktionen bereithält, um Schlüssel sicher zu generieren und zu verwalten.

Bei der AMD Secure Memory Encryption wird ein Schlüssel verwendet, um den gesamten Hauptspeicher der Maschine zu verschlüsseln. Beim Start generiert der AMD-SP diesen. Die Funktion muss im BIOS des Systems aktiviert werden und kann dann von jedem

[122] Vgl. Murdock, K. et al. (2020).
[123] Vgl. Intel Corporation (2020b).

Betriebssystem genutzt werden. Die technische Implementierung ist folgendermaßen gestaltet, wurde die Funktion aktiviert, wird das 48. Bit, Bit-Nummer: 47, da Zählung bei null beginnt, einer physischen Speicheradresse auf eins gesetzt, sodass diese Speicherseite dann als schützenswert markiert ist. Dieses Bit wird auch als C-Bit bezeichnet, dies wurde von *enCrypted* abgeleitet. Danach wird automatisch die AES-Engine genutzt zur Ver- und Entschlüsselung. Die folgende Grafik 8 verdeutlicht diesen Sachverhalt.

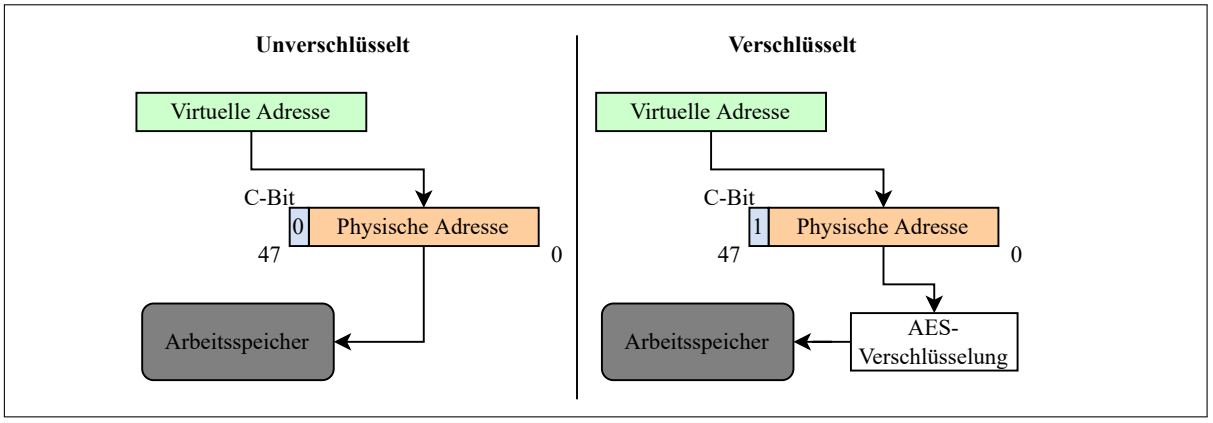


Abbildung 8: AMD-SME: Speicherverschlüsselung (Quelle: eigene Darstellung)

AMD SEV setzt für jede virtuelle Maschine und Hypervisor einen eigenen Schlüssel ein, sodass einzelne VMs voneinander isoliert werden, wobei hingegen bei AMD SME der gesamte Hauptspeicher verschlüsselt ist. Die Verwaltung der Schlüssel übernimmt dabei der AMD Secure Processor. Die Funktion muss vom Hypervisor sowie der einzelnen Gastbetriebssysteme unterstützt und aktiviert sein, bei der gesamten Verschlüsselung des Hauptspeichers, mit AMD SME, müssen der Hypervisor und das Betriebssystem keine spezielle Unterstützung dafür haben.[124]

SEV-ES Die Technologie SEV with Encrypted State (SEV-ES) ist eine Weiterentwicklung der vorangegangen SEV-Technik im Jahr 2017. AMD selbst schreibt, dass kein Sicherheitssystem 100 % sicher sein kann. SEV reduziert dabei schon einen großen Teil der Angriffsfläche. Nutzt beispielsweise eine VM aktiv Schlüssel, befinden sich diese im Speicher, aber auch in Registern der CPU selbst. Wird eine VM nicht richtig beendet, zum Beispiel durch einen Interrupt, werden die Registerzustände in den Speicher des Hypervisors gesichert und dieser ist frei zugänglich, sodass Daten gestohlen werden können oder verändert.

An dieser Stelle setzt die Weiterführung der SEV-Technik an, SEV-ES. Alle CPU-Register werden verschlüsselt und geschützt, sobald eine virtuelle Maschine beendet wird. Außerdem kann diese Technologie Veränderungen am Zustand von Registern feststellen und dann die Ausführung verhindern, sodass kein Schaden entstehen kann. Mittels AMD

[124] Vgl. Kaplan, D./Powell, J./Woller, T. (2021).

SEV-ES wird die Angriffsfläche weiter verkleinert und Schutz der VM gegenüber dem Hypervisor steigt.[125]

SEV-SNP AMDs neuste Technologie um virtuelle Maschinen sicher zu verwenden, trägt den Zusatz Secure Nested Paging (SNP) und wurde 2020 präsentiert. Die Erweiterung bietet neue Funktionen um die Integrität der Systeme zu wahren. Beispielsweise ist ein Ziel dieser Technologie Replay Angriffe zu verhindern, bei jenen Angriffen fängt der Angreifer die verschlüsselten Daten ab und setzt diese zu einem späteren Zeitpunkt wieder in den Speicher ein. Dadurch kann es zu Schäden innerhalb der Software aber auch des Speichers kommen. Die genutzte Software spielt dabei auch eine Rolle für den Ausgang des Angriffes, ob sie die fehlerhaften Daten als solche erkennt und dann verwirft.[126]

Die Integritätsprüfung geschieht folgendermaßen, dass eine virtuelle Maschine, welche Daten aus dem verschlüsselten Bereichs des Speichers liest, diese vor der Verwendung zunächst mit dem zuletzt in den gleichen Bereich geschriebenen Wert vergleicht. Stimmen beide Werte überein, ist die Integrität verifiziert und Daten können von der VM genutzt werden. Sind beide Werte hingegen ungleich, kann davon ausgegangen werden, dass es zu einer Veränderung kam und die CPU gibt eine Warnung aus. Die Markierung der abzusichernden Speicherseiten erfolgt ebenso durch das C-Bit, in dieser Version von SEV befindet es sich stets an der höchsten Stelle einer physischen Adresse. Die folgende Grafik 9 verdeutlicht diesen Sachverhalt.

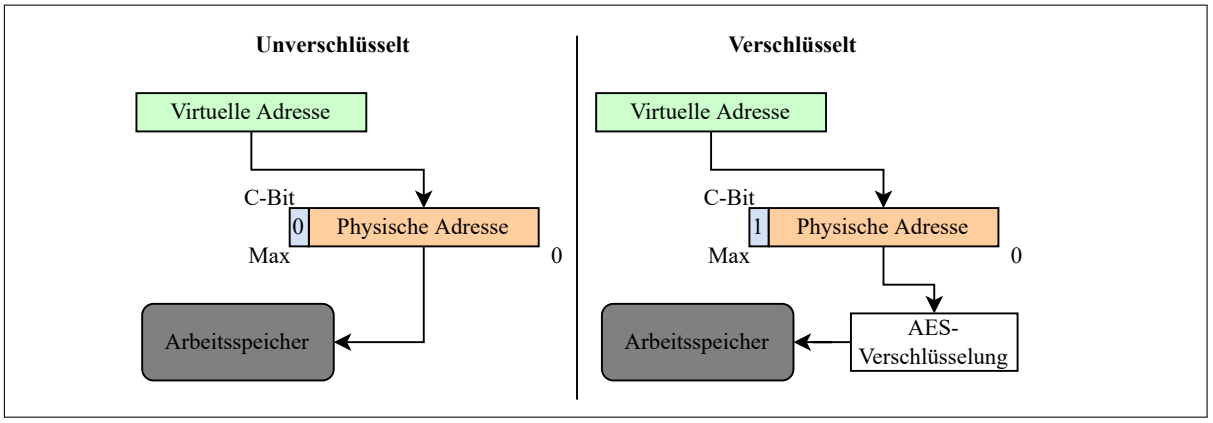


Abbildung 9: AMD-SEV-SNP: Speicherverschlüsselung (Quelle: eigene Darstellung)

Weitere Informationen und Spezifikationen der hier beschriebenen Technologien können dem von AMD veröffentlichten Architektur- und Programmierhandbuch entnommen werden.[127]

[125] Vgl. Kaplan, D. (2017).
[126] Vgl. Advanced Micro Devices, Inc. (2020).
[127] Vgl. Advanced Micro Devices, Inc. (2021).

6.2.2 Intel TDX

Im August 2020 veröffentlichte Intel ein eigenes White Paper zu ihrer neuen Technologie, den Trust Domain Extensions (TDX). Stand Juni 2022 hat Intel bereits elf verschiedene White Paper und Spezifikationen hinsichtlich Intel TDX veröffentlicht. Die Technologie wird erstmals mit der vierten Generation der XEON-Prozessoren eingeführt und soll ab der zweiten Jahreshälfte 2022 verfügbar sein.[128]

Bei dieser Technologie werden verschiedene Ansätze kombiniert, um virtuelle Maschine voneinander zu isolieren, die „Intel Virtual Machine Extensions (VMX) instruction-set-architecture (ISA) extensions, multi-key, total-memory-encryption (MKTME) technology, and a CPU-attested, software module“[129]. Die durch die Hardware-isolierten virtuellen Maschinen werden als Trust Domains bezeichnet. Das Ziel dieser Technik ist, dass kein Zugriff durch den Hypervisor auf die Daten der virtuellen Maschine möglich ist. Dieses Ziel umfasst zusätzlich die Möglichkeit der automatischen Skalierung der VM, was vor allem essentiell für die effiziente Nutzung von Ressourcen ist.

Durch die hardware-seitige Verschlüsselung des Hauptspeichers werden physische Angriffe auf das System verhindert. Weiterhin bietet die Technologie, ähnlich zu den Software Guard Extensions, die Möglichkeit der Remote Attestation, um dem Nutzer einer Anwendung die Möglichkeit zu geben, dass dieser, vor der Verwendung, die Integrität der Anwendung und der TDX-Plattform verifizieren kann.

Intel SGX-Enklaven können innerhalb einer Trusted Domain nicht ausgeführt werden. Die SGX-Technologie findet aber innerhalb von TDX Anwendung. So werden beispielsweise die **Quoting Enclaves** verwendet, um Bestätigungen der Infrastruktur durchzuführen und diese zu attestieren.[130]

Intel wird SGX und TDX nebeneinander auf Server-Plattformen anbieten, um vielseitige Möglichkeiten für die Nutzer bereitzustellen, wie Anil Rao, Vizepräsident der System Architektur und Entwicklungsabteilung von Intel, ankündigte: „With a full range of options including both Intel SGX and TDX, our customers can tailor their Confidential Computing solutions to their use cases' specific data isolation and deployment needs.“[131]

[128] Vgl. Intel Corporation (2022c).
[129] Intel Corporation (2020d), S. 1.
[130] Vgl. Intel Corporation (2021b).
[131] Rao, A. (2022).

6.2.3 ARM TrustZone

ARM-Prozessoren werden vordergründig in mobilen Geräten oder Geräten mit speziellen Einsatzzwecken verwendet. Diese Prozessoren werden für ihr Einsatzgebiet spezifisch entwickelt. TrustZone ist eine Technik welche in die Hardware der Mikrocontroller implementiert wird. Sie wurde bereits Ende der 2000er Jahre eingeführt. Sie unterteilt dabei die Ausführungsumgebung samt des Speicher, der externen Geräte und die Funktionen in sicher und nicht sicher. Eine sogenannte Speicherschutzzeit wird jeder Umgebung zur Verfügung gestellt, sodass Speicherregionen isoliert werden können.[132]

Bei der Entwicklung eines eingebetteten Systems müssen die Entwickler es in mindestens zwei verschiedene Projekte teilen, einmal das Projekt welches zur sicheren Ausführung gedacht ist, es wird meist mit Firmware-Projekt betitelt und einmal das Projekt zur nicht sicheren Ausführung, oft Benutzerprojekt genannt.

Beim Start eines Mikrocontrollers, mit aktivierter TrustZone-Funktionalität, beginnt dieser im abgesicherten Zustand und startet daraufhin das eigentliche System. Der Mikrocontroller wechselt danach in den nicht sicheren Bereich, damit er die Anwendung des Nutzers ausführen kann.

Die Benutzeranwendung kann nur mittels eines sicheren Gateways auf Funktionen und Speicher zugreifen. Versucht die Anwendung ohne dieses Zugriff auf den Speicher zu bekommen, wird eine Warnung ausgelöst.

Die folgende Abbildung 10 zeigt den beschriebenen Aufbau.

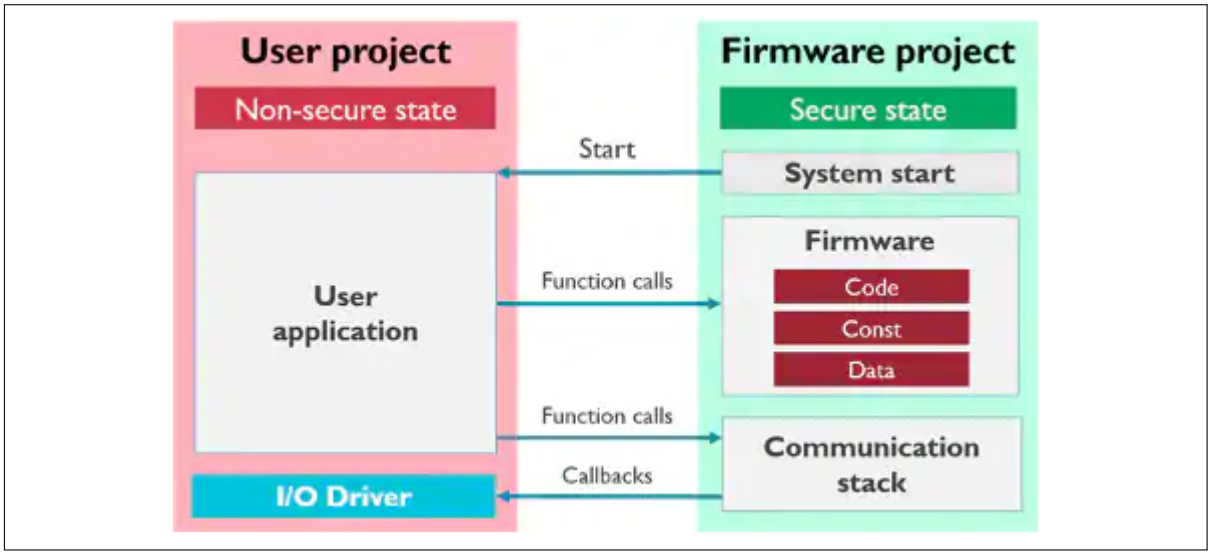


Abbildung 10: Aufteilung einer Anwendung in zwei Projekte, um eine Isolierung zu erreichen (Quelle: Beningo, J. (2020))

[132] Vgl. ARM Limited (2009).

Tiefgreifendere Informationen können dem durch ARM veröffentlichten Handbuch zu TrustZone entnommen werden.[133]

[133] Vgl. ARM Limited (2009).

7 Absicherung der Vault Software mittels Confidential Computing

In diesem Abschnitt der Arbeit soll die bestehende Open-Source Lösung HashiCorp Vault als Schlüssel- und Geheimnisverwaltungssoftware genutzt werden. Um die Ausführung der Anwendung vor Zugriffen des System-Besitzers zu schützen soll Confidential Computing eingesetzt werden, in diesem konkreten Fall Intel SGX.

Zunächst wird der Aufbau und das Konzept der Umsetzung vorgestellt, mit den verwendeten Komponenten. Danach wird aufgezeigt, wie die praktische Umsetzung dessen abläuft. Abschließend wird eine Auswertung der Umsetzung vorgenommen.

7.1 Implementierung des Konzepts

Damit die Nutzung effizient und flexibel ist, wird auf Cloud-Infrastrukturen gesetzt, dadurch können schnell und einfach neue Ressourcen hinzugefügt werden bei Bedarf. Da im Unternehmen bereits die Microsoft Azure Cloud genutzt wird für Projekte und Testzwecke, wird sie auch in dieser Arbeit genutzt. Durch die Verwendung dieser Cloud-Umgebung besitzt man die Hardware nicht und muss dem Anbieter, Microsoft, vertrauen, dass dieser sie nicht missbräuchlich nutzt.

An dieser Stelle kommt Confidential Computing zum Tragen, dadurch kann die Verarbeitung von Daten und die Ausführung von Anwendungen auf entfernten System abgesichert werden. Microsoft als Cloud-Anbieter bietet zwei verschiedene Herangehensweisen für TEE, die vertrauenswürdigen Ausführungsumgebungen, an, zum Einen Intel Prozessoren, welche SGX unterstützen und zum Anderen AMD Prozessoren mit der SEV-Technologie. Beide Prozessoren können in virtuellen Maschinen verwendet werden. Zudem können die Intel CPUs genutzt werden im eigenen Kubernetes Service von Azure, sodass Anwendungen innerhalb des Clusters die SGX-Funktionalität nutzen können.

Bei der Verwendung des AMD-Ansatzes ist der gesamte Hauptspeicher einer virtuellen Maschine verschlüsselt, dies geschieht durch spezielle Bestandteile der CPU, erfordert aber auch eine softwareseitige Unterstützung durch den Hypervisor. Hingegen bei SGX nur der Speicher einer Enklave abgesichert ist. Damit eine Anwendung SGX verwenden kann, ist eine Anpassung des Codes notwendig, bei AMD SEV kann die Anwendung direkt unverändert in der VM ausgeführt werden. Der Nutzer kann die SGX Plattform als solches durch die Nutzung des Intel Services attestieren lassen und so sichergehen, dass die Integrität gegeben ist. Bei AMD attestiert ein Teil der CPU die Plattform.

Ziel der Umsetzung ist die Anwendung möglichst klein und portabel zu halten, sodass das Container-Konzept verwendet wird. Damit wird der *Cloud-Native* Ansatz verfolgt und es wird keine vollwertige und komplexe VM benötigt. Außerdem erfordert eine VM auch das Vertrauen in den Hypervisor selbst. Deshalb fällt die Wahl der Trusted Execution Environment auf die Technologie Intel SGX samt Enklaven.

In dieser Arbeit wird wie angesprochen das Container-Konzept verfolgt, unter Nutzung von Docker-Containern. Diese Container sind die laufenden Instanzen eines Docker-Images. Images sind Dateien, welche nur gelesen werden können, die Container hingegen sind aktiv, haben ausführbaren Inhalt und sind kurzlebig. Einige Vorteile von Docker-Containern sind für unsere Umsetzung besonders relevant:

- **Isolierte Umgebung** Durch die isolierte und klar definierte Umgebung, kann sich während der Umsetzung komplett auf die Migration der Anwendung selbst konzentriert werden.
- **Mobilität** Die Docker Images und die daraus entstehenden Container verhalten sich auf unterschiedlichen Plattformen gleich.
- **Flexibilität** Muss ein Paket oder eine Bibliothek innerhalb des Images aktualisiert werden, kann dies schnell und einfach geschehen. Anschließend getestet werden und bei Erfolg ausgerollt.

In dieser Arbeit werden die entstehenden Container Images in den Docker Hub geladen, um sie zu nutzen und verteilen zu können. Dadurch ist keine Abhängigkeit von einem Hersteller und deren Image Registry gegeben, wie beispielsweise die Azure Container Registry.

Es wird eine Orchestrierungssoftware benötigt, um mehrere Container gleichzeitig zu verwalten. Sie kann Container in großer Anzahl steuern, überwachen und planen. Die zwei wohl bekanntesten Vertreter sind Docker Swarm und Kubernetes. In dieser Arbeit wird Kubernetes zur Orchestrierung der Container gewählt, da es die Funktion zum automatischen Skalieren bietet, integriertes Logging und Monitoring hat, sowie am häufigsten genutzt wird. Es ist Open-Source und wird unter der Cloud Native Computing Foundation weiterentwickelt.

Kubernetes kann auf der Hardware selbst durch den Nutzer installiert werden, dieser muss dann eine Verbindung zwischen den einzelnen Knoten sicherstellen und ist verantwortlich für die komplette Konfiguration. Alle bekannten Cloud-Anbieter haben eigene Implementierungen von Kubernetes, welche ein fertig konfiguriertes Cluster dem Nutzer zur Verfügung stellen. Wie bereits angesprochen wird in dieser Arbeit die Azure Cloud Plattform genutzt werden, sodass der angebotene Azure Kubernetes Service verwendet

wird. Er ist auch Open-Source und wurde im Jahr 2018 erstmals veröffentlicht. Zudem bietet er die Möglichkeit Knoten mit SGX-Unterstützung zu verwenden.

Um komplexe Anwendungen mittels Kubernetes auszuführen, ist es notwendig verschiedene Kubernetes Ressourcen zu verwenden, wie beispielsweise Pods, Services oder Deployments. Jede Ressource muss manuell angelegt und konfiguriert werden. An dieser Stelle vereinfacht ein Paket-Manager die Arbeit erheblich, dieser unterstützt bei der Installation und Aktualisierung von Anwendungen. In der Kubernetes Umgebung ist der durch die Cloud Native Computing Foundation betreute Helm Paket-Manager der Standard und wird deshalb auch in dieser Arbeit genutzt.[134]

Die notwendigen Konfigurationsdateien und Dateien für den Paketmanager werden in eigenen Repositories auf GitHub gespeichert, dadurch ist auch möglich Unterschiede zwischen verschiedenen Versionen der Umsetzung nachvollziehen zu können. Bei Fehlern kann auf eine ältere, funktionierende Lösung gewechselt werden.

Die automatische Erstellung und Bereitstellung von Infrastruktur wird immer bedeutsamer, dadurch dass der Arbeitsaufwand für Administratoren erheblich sinkt. Außerdem muss die Infrastruktur flexibel sein und mitwachsen, ähnlich wie andere Cloud-Strukturen. Der recht neu entstandene Begriff GitOps beschreibt die Möglichkeit, dass Entwickler Aufgaben übernehmen, welche normalerweise von einem separaten IT Team übernommen werden. Außerdem wird Git genutzt, um die Verwaltung und Entwicklung von Kubernetes Ressourcen zu ermöglichen. GitOps Tools synchronisieren die Kubernetes Infrastruktur mit Konfigurationsdateien in einem Git Repository. Dadurch kann auch leicht zwischen verschiedenen Versionen gewechselt werden. Ein weitere Vorteil ist die Visualisierung der Infrastruktur. Zwei bekannte Vertreter von GitOps Tools sind zum einen Argo CD und zum anderen Jenkins X. Jenkins X erfordert eine größere Einarbeitung und ist unter anderem deswegen nicht so bekannt und etabliert wie Argo CD. Die Wahl des GitOps Tools fiel in dieser Arbeit deshalb auf Argo CD. Beide Lösungen funktionieren mit Kubernetes und sind darauf abgestimmt.[135]

Im Abschnitt 6.1.8 wurden verschiedene Herangehensweisen zur Migration einer Anwendung zur Nutzung von SGX beschrieben, weiter wurden vorhandene wissenschaftliche Studien, aber auch proprietäre Lösungen vorgestellt. Bei der Migration ist es essentiell zu unterscheiden, welche Teile der Anwendung vertrauenswürdig sind und welche nicht, sodass entschieden werden kann, was innerhalb der Enklave ausgeführt werden muss, um die Vertraulichkeit zu wahren. Einige, vor allem kommerzielle Lösungen, wählen einen schnelleren Weg und führen die Anwendung, welche in einer höheren Programmiersprache, wie zum Beispiel Python oder Java, geschrieben ist, direkt als Container aus und

[134] Vgl. Helm Authors (2022).
[135] Vgl. Argo Project Authors (2022).

nehmen keine Änderungen am eigentlichen Code vor. Firmen, wie Scontain oder Fortanix verfolgen diesen Ansatz, zudem bietet beispielsweise Scontain auch schon fertig migrierte Container-Images an.

Das Ziel dieser Arbeit ist aber die Nutzung von Open-Soure-Lösungen, um auch so volle Kontrolle über den Code innerhalb des Containers zu haben. So wurden im Abschnitt 6.1.8 verschiedene Ansätze gezeigt, welche quelloffen sind. So kann ein bestehendes SDK, wie das Intel SGX SDK, verwendet werden, um Programme, welche in C geschrieben sind, zur Ausführung in einer Enklave zu bringen. Liegt beispielsweise der Programmcode in Java oder Node vor, so kann Graphene-SGX als Migrationsansatz gewählt werden. Es ist auch sehr gut dokumentiert.

In Fall von HashiCorp Vault ist die Software in der Programmiersprache Go geschrieben, sodass das Intel SGX SDK oder Graphene-SGX nicht passen. Das Open-Source-Projekt EGo der Firma Edgeless Systems aus Deutschland basiert auf dem, in dem Gebiet von Confidential Computing und Trusted Exection Environments, standardisierten Open Enclave SDK und ist auf die Programmiersprache Go ausgerichtet. Weiterhin bietet diese Lösung zusätzliche Funktionen wie Sealing oder Remote Attestation an. Im Zuge dieser Arbeit wird EGo zur Migration der Anwendung verwendet.

Um die Anwendung und die Infrastruktur zu überwachen werden Werkzeuge benötigt, welche zunächst die Daten erheben und anschließend verarbeiten, auswerten und grafisch darstellen können. Prometheus wird genutzt, um die Daten von der Kubernetes Infrastruktur zu gewinnen und zu speichern. Außerdem veröffentlicht der Dienst die Daten an eine festgelegte Schnittstelle.[136] Grafana als Visualisierungswerkzeug kann diese Schnittstelle nutzen und die Daten abfragen. Anschließend kann es die erhaltenen Metriken grafisch darstellen, auch über einen gewissen Verlauf der Zeit. Mittels dieser Überwachungssoftware können Probleme innerhalb der Infrastruktur und genutzten Container erkannt werden und anschließend behoben werden.[137]

Damit die Dienste auch von außerhalb erreicht und verwendet werden können, wird ein Reverse Proxy eingesetzt. Ohne die Verwendung dessen kann entweder per IP und Port auf einen bereitgestellten Service zugegriffen werden von außen oder mit den Kubernetes Werkzeugen lokal zeitweise freigegeben werden. Dabei verbindet man sich zunächst mit dem Kubernetes Cluster und leitet einen Service lokal an einen Port weiter, sodass nur der Zugriff vom lokalen Gerät aus geschehen kann.

Ziel dieser Arbeit ist, dass Vault, sowie die Oberflächen von Argo CD, Grafana und die vom Reverse Proxy selbst, über eine Domain angesprochen werden können, von überall. Jeder Dienst erhält eine eigene Subdomain, welche alle auf die gleiche IP des Reverse Proxys

[136] Vgl. Prometheus Authors (2022).
[137] Vgl. Grafana Labs (2022).

zeigen, dieser übernimmt die Unterscheidung und leitet die Anfragen an den richtigen Service weiter.

In dieser Arbeit wird Traefik als Reverse Proxy in Kubernetes eingesetzt, er überzeugt mit seiner sehr guten Unterstützung von Kubernetes und vergleichsweise leichten Konfiguration.[138] Er ist ein bewährter Reverse Proxy für Microservice-Umgebungen. In Cloud- und Containerumgebungen kann sich schnell die Anzahl und Verfügbarkeit der Container ändern, darauf wurde Traefik optimiert, um stets alle entsprechenden Routingziele zu erfassen. Er unterstützt verschiedenste Protokolle und Routing Muster, wie zum Beispiel Load Balancing. Traefik ist als Open-Source Version verfügbar und wurde in Go geschrieben. Er unterstützt zudem unterschiedliche Umgebungen und kann so automatisch Services entdecken und die Konfiguration für diese ableiten. Zusätzlich übernimmt er die Secure Sockets Layer (SSL)-Zertifikatsverwaltung und kann automatisch Zertifikate bei LetsEncrypt anfordern, außerdem kann umfassendes Logging konfiguriert werden.

Zur Absicherung der Verbindungen zu den Oberflächen und Vault selbst wird HTTPS mittels Zertifikaten eingesetzt, der Reverse Proxy übernimmt die Verwaltung dieser. In dieser Arbeit werden Zertifikate der CA Let's Encrypt verwendet. Sie bieten folgende Vorteile[139]:

- **Kostenlos** Jeder Eigentürmer einer Domain kann Zertifikate kostenlos anfordern.
- **Automatisch** Viele bestehende Anwendungen können selbstständig Zertifikate beantragen und bei Bedarf auch erneuern.
- **Transparent** Alle veröffentlichen und zurückgezogenen Zertifikate werden öffentlich zugänglich gemacht, sodass jeder diese inspizieren kann.
- **Open-Source** Die Protokolle der automatischen Ausstellung und Erneuerung wurden veröffentlicht als offener Standard, sodass er von anderen adoptiert werden kann.

Die erstellten Zertifikate haben stets eine Gültigkeit von 90 Tagen. Das Vertrauen in die Zertifikate besteht durch die Zertifikatskette mit Zwischenzertifikaten und dem Wurzelzertifikat der Internet Security Research Group (ISRG), dem *ISRG Root X1*. Alle heutigen Plattformen und Browser, wie zum Beispiel iOS, Android oder Mozilla Firefox vertrauen diesem Wurzelzertifikat. Let's Encrypt nutzt verschiedene Limits, um einen Missbrauch ihres Dienstes zu verhindern. Beispielsweise können maximal 50 Zertifikate für eine registrierte Domain pro Woche angefordert werden oder fünf fehlgeschlagene Validierungen pro Domain pro Stunde. Dieses Limit kann gerade bei Testumgebungen schnell erreicht

[138] Vgl. Traefik Labs (2022).
[139] Vgl. Let's Encrypt (2022a).

werden. Um diesen Limits zu entgehen, setzt Let's Encrypt ein sogenanntes Staging Environment ein, bei diesem sind die Limits deutlich erhöht.[140] Die in dieser Umgebung erstellten Zertifikate werden von anderen Root-Zertifikaten abgeleitet, welche nicht in den Zertifikatsspeichern von Browsern oder anderen Plattformen vorhanden sind. Dadurch werden Browser zum Beispiel eine Warnung ausgeben, wenn versucht wird, auf eine Seite zuzugreifen, aber anhand der Namen und Aussteller der Zertifikate kann sichergestellt werden, dass der Prozess der Ausstellung der Zertifikate und Verwendung im Server problemlos funktioniert. Ist der Test erfolgreich gewesen, kann in die Produktivumgebung von Let's Encrypt gewechselt werden, dies geschieht mittels Änderung der URL, welche für die Zertifikatsanfragen genutzt wird.

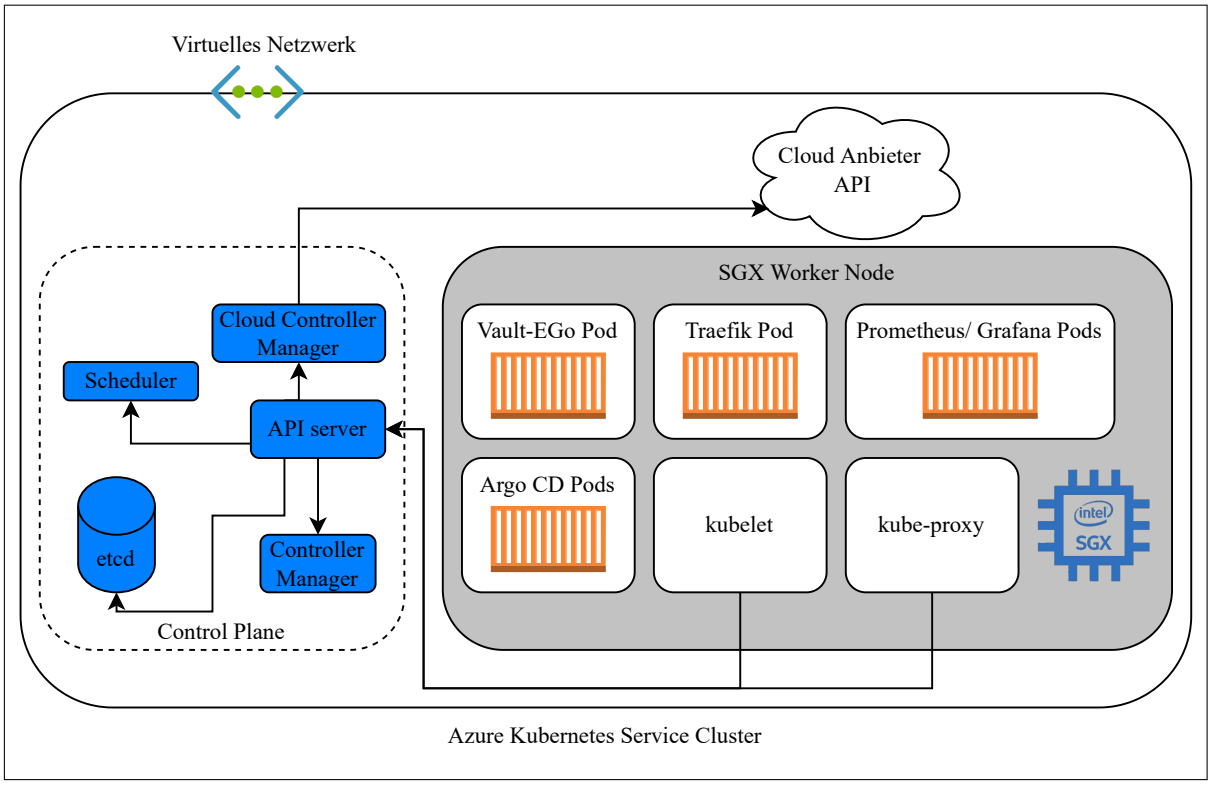


Abbildung 11: Struktur des Konzeptes (Quelle: eigene Darstellung)

Die Abbildung 11 zeigt zusammenfassend die Struktur des Konzeptes. Es wird ein Azure Kubernetes Service Cluster, mit SGX-Unterstützung, verwendet. HashiCorp Vault wird mittels EGo SGX-kompatibel gemacht und in einem Container zur Ausführung gebracht. Dieser Container wird im Kubernetes Cluster ausgeführt. Prometheus und Grafana werden für das Logging und Monitoring genutzt. Traefik, mit Let's Encrypt Zertifikaten, kommt zum Einsatz als Reverse Proxy, um die einzelnen Oberflächen abgesichert erreichbar zu machen. Mittels Argo CD und GitHub wird die Struktur des Clusters und der Anwendungen definiert, um die Entwicklung zu vereinfachen und um diese zu reproduzieren.

[140] Vgl. Let's Encrypt (2022b).

7.2 Praktische Umsetzung

In diesem Abschnitt der Arbeit werden die durchgeführten Schritte beschrieben, um das vorher beschriebene Konzept umzusetzen. Die verwendeten Konfigurationsdateien sind im Anhang fünf bis 19 zu finden. Um die Fehlersuche zu vereinfachen, wurden die Teile des Konzepts zunächst einzeln umgesetzt und nach erfolgreicher Umsetzung mit dem nächsten Schritt begonnen, sodass Probleme im vorherigen Schritt nahezu auszuschließen sind.

Der wichtigste Schritt ist die Migration der bestehenden Vault Anwendung zu einer SGX-fähigen Anwendung. In dieser Arbeit wird dafür EGo verwendet, als Migrationswerkzeug, da HashiCorp Vault in der Programmiersprache Go geschrieben ist.

In dieser Arbeit wird eine virtuelle Maschine mit SGX-Unterstützung in der Azure Cloud genutzt. Die Maschinen der DCsv2-Familie haben diese Funktionalität. Die konkrete Instanz ist eine Standard_DC2s_v2 mit folgenden Eigenschaften: 2 physischen Kernen, 8 GB Arbeitsspeicher und 100 GB SSD Speicher und 56 MB Enclave Page Cache. Es wird ein Linux Betriebssystem verwendet, genauer gesagt Ubuntu 18.04 in der LTS Version. Azure bietet virtuelle Maschinen in zwei verschiedenen Generationen an, in dieser Arbeit wird die neuere, zweite Generation verwendet, da nur diese die Unterstützung von SGX haben. Die Abbildung 12 zeigt einige Eigenschaften der verwendeten Maschine.

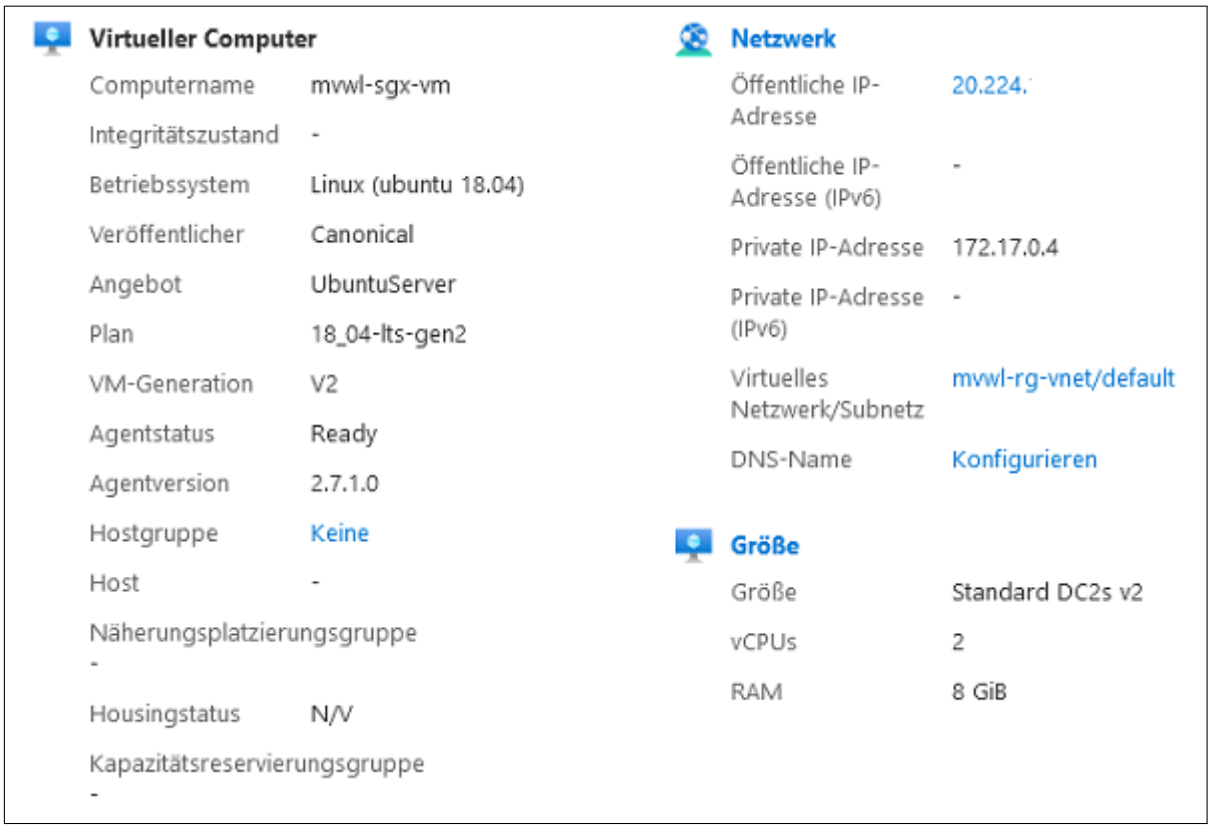


Abbildung 12: Virtuelle Maschine mit SGX-Unterstützung in Azure (Quelle: eigene Darstellung)

Zunächst wurde auf der virtuellen Maschine Go installiert, um das normale Kompilieren der Vault Anwendung zu testen. Dafür wurde Go über das angebotene Paket installiert. Der Sourcecode von Vault wurde dann von GitHub auf die virtuelle Maschine geklont und in das Verzeichnis gewechselt. Als Erstes wurden benötigte zusätzliche Werkzeuge geladen, um die Umgebung vorzubereiten. Anschließend konnte eine lauffähige Binary erzeugt werden. Diese wurde ausgeführt und die Vault Anwendung wurde kurz getestet. Da dieser Test erfolgreich verlaufen ist, wird als nächstes Vault mittels EGo kompiliert.

Für die Nutzung der SGX-Funktionalitäten sind einige zusätzlichen Pakete von Intel notwendig. Das Advanced Packaging Tool (APT)-Repository wird hinzugefügt, daraufhin können alle benötigten Pakete installiert werden. Daraufhin kann das EGo Debian Paket heruntergeladen und installiert werden. Nach erfolgter Installation wird die Funktionalität EGo's SGX zu nutzen getestet, verwendet wird dafür ein vorgefertigtes Beispiel, eine Enklaven-Anwendung, welche fünfmal die Worte *hello* und *world* auf der Kommandozeile ausgibt. Dafür wird zunächst der benötigte Go-Code kompiliert, die daraus entstandene Binary wird signiert und anschließend kann diese ausgeführt werden. Ist dies erfolgreich geschehen, ist EGo einsatzfähig und SGX kann verwendet werden.

Nachdem die herkömmliche Erstellung einer Vault Binary erfolgreich war und die Funktionalität von EGo und SGX verifiziert wurde, geht es nun zur Migration von Vault. Dafür wird in einen leeren Ordner gewechselt und erneut der Quellcode von Vault von GitHub geladen. Nachdem dies abgeschlossen ist, kann EGo die Binary erzeugen. Das Kompilieren einer solch umfangreichen Anwendung benötigt etwas Zeit. Nach erfolgreicher Erstellung der Binary wird diese noch signiert. Anschließend erhält man eine Vault Binary, welche die Funktionen einer SGX-Enklave nutzt und dadurch abgesichert ist. Mittels EGo kann die Binary ausgeführt werden, durch die Verwendung des *ego run*-Befehls. Der Binary können ganz normal Parameter und Optionen übergeben werden.

An dieser Stelle wird der Vault Server gestartet mit gewissen Parametern, um diesen zu testen. Er läuft daraufhin im Vordergrund. Durch eine zweite Verbindung zur virtuellen Maschine kann die Funktionalität des Vault Servers überprüft werden, dafür kann auf der Seite des Clients eine gewöhnliche Binary genutzt werden. Kann der Client sich erfolgreich zum Server verbinden und mit diesem interagieren, ist der Test abgeschlossen und damit die Migration der Vault Anwendung.

Ziel der Arbeit ist die abgesicherte Ausführung von Vault in einem Kubernetes Cluster, dafür muss Vault innerhalb eines Docker Containers ausgeführt werden. Nach erfolgreicher Erstellung einer SGX-fähigen Binary auf einer virtuellen Maschine muss diese im Container zur Ausführung gebracht werden. Dafür muss das zugrundeliegende Image und die zusätzlich benötigten Komponenten bestimmt werden. Ein Vorteil von Docker ist die Erstellung einer einzigen Datei, um die Umgebung zur Ausführung vorzubereiten und zu

konfigurieren, das Dockerfile. Es ist eine einfache Textdatei, welche Anweisungen enthält zur Erstellung eines Images. Vor der Erstellung des Dockerfiles und damit dem Image wird versucht die Vault Binary innerhalb einem normalen Container zu erstellen und auszuführen.

Als zugrundeliegendes Image wird Ubuntu 18.04 gewählt. Zunächst wird damit ein interaktiver Container gestartet, zusätzlich wird das SGX-Gerät des Host-Systems in den Container gemountet, damit dieser auch die SGX-Funktionalitäten nutzen kann und der Port 8200 geöffnet, um im Anschluss die Funktionalität des Vault Servers überprüfen zu können. Nach erfolgter Erstellung kann mittels des *docker* Kommandozeilenwerkzeugs eine Shell im Container gestartet werden, daraufhin wechselt man in den Container. Anschließend kann die SGX-Funktionalität überprüft werden. Ist das Vorhandensein des SGX-Gerätes sichergestellt, kann die Nutzung von EGo vorbereitet werden. Dafür werden gewisse Pakete benötigt, zusätzlich auch welche aus einem Intel APT-Repository, sodass dieses noch zu den verfügbaren Paketlisten hinzugefügt werden muss. Anschließend kann EGo installiert werden, durch das fertige Debian Paket, welches zunächst heruntergeladen werden muss von GitHub.

Wurde EGo erfolgreich installiert, wird der Quellcode der Vault Software von GitHub geklont. Anschließend wechselt man in das entsprechende Verzeichnis und erstellt die Vault Binary. Nach erfolgter Kompilierung muss diese noch signiert werden. Anschließend kann die Binary mit den bekannten Parametern zur Testung ausgeführt werden. Nach dem erfolgreichen Start des Servers kann erneut eine zweite Verbindung zur virtuellen Maschine hergestellt werden, um mit einer normalen Vault Binary den Server innerhalb des Containers zu testen. Ist auch dieser Test erfolgreich verlaufen, kann Vault unter Nutzung von SGX innerhalb eines Containers ausgeführt werden.

Nachdem die benötigten Pakete identifiziert wurden, welche auf einem Ubuntu 18.04 Docker Image nachinstalliert werden müssen, kann zur Erstellung des Dockerfiles übergangen werden. Es enthält unter anderem Kommandos, welche ein Anwender auf der Kommandozeile ausführen kann, um selbstständig ein Image zu erstellen. Die folgende Abbildung 13 zeigt das entwickelte Dockerfile, außerdem ist es im Anhang fünf zu finden.

```
FROM ubuntu:18.04 AS base

WORKDIR /

RUN apt update -y && apt upgrade -y
RUN apt install -y wget git build-essential software-properties-common libssl-dev

RUN wget -qO- https://download.01.org/intel-sgx/sgx_repo/ubuntu/intel-sgx-deb.key | apt-key add
RUN add-apt-repository "deb [arch=amd64] https://download.01.org/intel-sgx/sgx_repo/ubuntu `lsb_release -cs` main"

RUN wget https://github.com/edgelesssys/ego/releases/download/v0.5.0/ego_0.5.0_amd64.deb
RUN apt install -y ./ego_0.5.0_amd64.deb

RUN git clone https://github.com/hashicorp/vault

EXPOSE 8200

ADD script.sh /
RUN chmod +x /script.sh
CMD ./script.sh
```

Abbildung 13: EGo-Vault: Dockerfile (Quelle: eigene Darstellung)

Zunächst wird mittels der *FROM*-Anweisung spezifiziert, welches Image als Grundlage genutzt werden soll. In diesem Fall Ubuntu 18.04, dieses ist der Startpunkt für das fertige Image. Anschließend wird das Verzeichnis festgelegt, in welchem begonnen wird zu arbeiten. Danach werden die Paketlisten aktualisiert und alle Pakete, für welche eine Aktualisierung verfügbar ist, aktualisiert. Nachdem dies erfolgt ist, werden die zuvor identifizierten Pakete installiert, um EGo zu installieren und Vault damit zu migrieren. Um die SGX-Funktionalität zu verwenden, wird das Intel Repository, samt seines Schlüssels, hinzugefügt. Daraufhin wird das EGo Debian Paket heruntergeladen und installiert. Bei der Installation werden zusätzliche benötigte Komponenten aus dem Intel Repository geladen. Damit ist die Umgebung zur Migration vorbereitet und es kann der Quellcode von Vault in das Arbeitsverzeichnis geklont werden. Standardmäßig verwendet der Vault Server den Port 8200 für Anfragen. Mittels der *EXPOSE*-Anweisung wird Docker informiert, dass der Container auf diesem Port lauscht.

Zusätzlich wird ein Skript genutzt, um HashiCorp Vault mittels EGo zu kompilieren und anschließend auszuführen. Dieses wird mittels der *ADD*-Anweisung in das Dateisystem des Images kopiert. Danach wird ds ausführbar gemacht. Die *CMD*-Anweisung stellt einen Standard für den ausgeführten Container dar, beispielsweise die Ausführung eines Programms oder Befehls. In diesem Fall wird das Skript ausgeführt. Die Abbildung 14 zeigt das genutzte Skript, zusätzlich ist es im Anhang sechs abgelegt.

```
#!/bin/bash

cd vault
ego-go build -v -o bin/vault
cd bin
ego sign vault
ego run vault server -dev -dev-no-store-token -dev-root-token-id mytoken -dev-listen-address="0.0.0.0:8200"
```

Abbildung 14: EGo-Vault: Skript (Quelle: eigene Darstellung)

Im Folgendem wird das einfache Shell-Skript kurz erläutert. Zunächst wird in den geklonten Ordner des Vault Sourcecodes gewechselt. Danach wird die Binary mittels EGo kompiliert und anschließend signiert. Zu guter Letzt wird die Binary ausgeführt und der Vault Server mit bestimmten Parametern gestartet. Der Server wird im Entwicklungsmodus gestartet, dieser ist nicht für den Produktiveinsatz geeignet, da Vault automatisch entsiegelt ist.

Nachdem das Dockerfile und das Skript vorbereitet sind, kann das Image lokal gebaut werden. Nach der erfolgreichen Erstellung kann dieses Image getestet werden, in dem es ausgeführt wird, mit gemounteten SGX-Gerät und geöffneten Port 8200. Startet der Container ohne Probleme, kann erneut mittels einer herkömmlichen Vault Binary der Server überprüft werden. Die Abbildung 15 zeigt den erfolgten Build Prozess des Images.

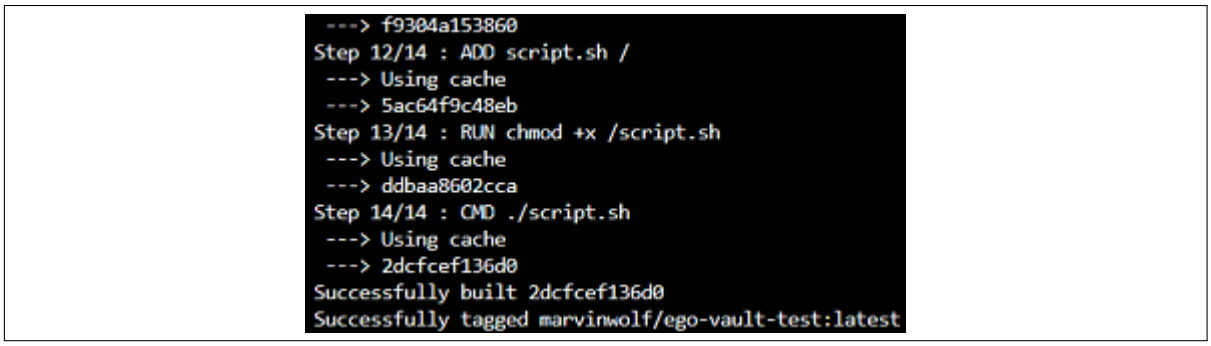


Abbildung 15: EGo-Vault: Image Build Prozess (Quelle: eigene Darstellung)

Ist dieser Test auch erfolgreich verlaufen, wird das fertige Image zum Docker Hub hochgeladen, damit dieses Image öffentlich zugänglich ist und später im Kubernetes Cluster genutzt werden kann. Die folgende Abbildung 16 zeigt das hochgeladene Image zum Docker Hub.

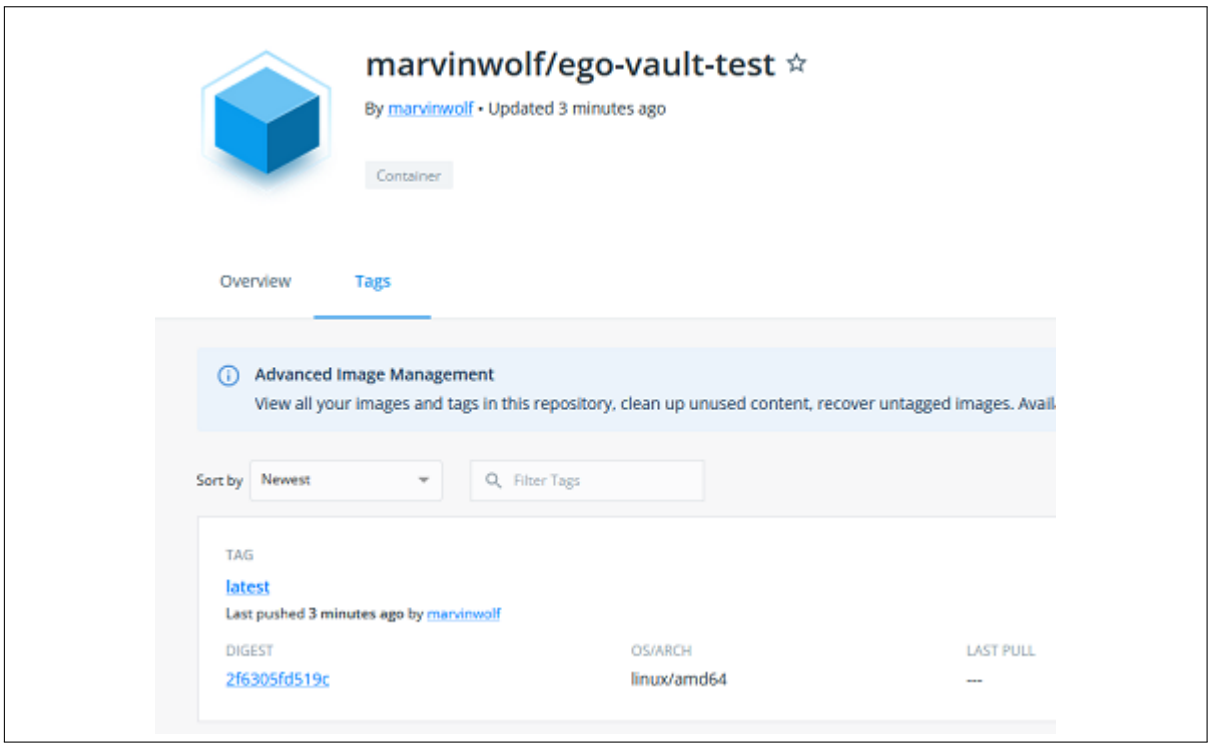


Abbildung 16: EGo-Vault: Image im Docker Hub (Quelle: eigene Darstellung)

An dieser Stelle existiert ein fertiges Image, welches zur Ausführung von HashiCorp Vault unter Nutzung von Intel SGX, genutzt werden kann. Um Applikationen oder Services in Kubernetes auszurollen, ist es notwendig manuell Kubernetes Konfigurationsdateien zu erstellen. Helm als Paket-Manager in Kubernetes erleichtert diesen Prozess. Die sogenannten Helm Charts definieren die Anwendungen oder Ressourcen, welche erstellt werden sollen. Sie helfen beim installieren und aktualisieren. Für die migrierte Vault Anwendung wird so eine Helm Chart (Anhang 7) angelegt, sie enthält zum Einen das Deployment (Anhang 8) der Anwendung selbst und zum Anderen den Service (Anhang 9), welcher den Port 8200 der Anwendung zur Verfügung stellt. Innerhalb des Deployments wird das zu verwendende Image angegeben, in diesem Fall das zuvor auf den Docker Hub Hochgeladene. In dieser Arbeit soll der Azure Kubernetes Service genutzt werden. Durch das Hinzufügen eines Ressourcen Limits des nutzbaren Enclave Page Caches, wird dieses Deployment automatisch auf einem Knoten im Kubernetes Cluster platziert, welcher SGX unterstützt, zusätzlich wird das benötigte SGX-Gerät gemountet. Die Helm Chart wird noch gebaut und anschließend zu GitHub hochgeladen, damit diese auch für das Cluster zugänglich ist.

Als nächster Schritt der Umsetzung steht die Erstellung eines Kubernetes Clusters mittels des Azure Kubernetes Service an. Es wird ein Cluster mit einem Knoten angelegt, dieser Knoten stammt auch aus der DCsv2-Familie, somit bringt er auch SGX-Unterstützung mit sich. Es handelt sich um eine Standard_DC4s_v2 Instanz mit den folgenden Eigenschaften: 4 physische Kerne, 16 GB Arbeitsspeicher und 112 MB nutzbaren Enclave Page

Cache. Zusätzlich wird bei der Erstellung angegeben, dass das Add-On *Confcon* genutzt werden soll, es ist das Plugin von Azure zur Nutzung von Intel SGX, sodass der EPC innerhalb des Clusters genutzt werden kann. Die Abbildung 17 zeigt den entsprechenden Befehl, um solch ein Azure Kubernetes Service Cluster zu erstellen.

```
az aks create --resource-group mvwl-rg-sgx --name AKS-SGX --node-vm-size Standard_DC4_v2 \
--node-count 1 --enable-addon confcom \
--network-plugin azure --vm-set-type VirtualMachineScaleSets --aks-custom-headers usegen2vm=trueen2vm
```

Abbildung 17: Azure Kubernetes Service Cluster: Erstellung (Quelle: eigene Darstellung)

Nach erfolgter Erstellung kann durch die Kontrolle der aktuell laufenden Pods im Kubernetes Cluster überprüft werden, ob die SGX-Funktionalität vorhanden ist. An dieser Stelle hat man ein Kubernetes Cluster in Azure mit einem Knoten und der Unterstützung von SGX. Die Abbildung 18 zeigt die entsprechenden Pods im Cluster.

```
kube-system   sgx-plugin-v9ljn                    1/1     Running   0          2d8h
kube-system   sgx-webhook-f4b55fb9f-rxjx4         1/1     Running   0          2d8h
```

Abbildung 18: Azure Kubernetes Service Cluster: SGX-Pods (Quelle: eigene Darstellung)

Mit dem frisch erstellten Kubernetes Cluster kann die erstellte Helm Chart der abgesicherten HashiCorp Vault Anwendung getestet werden. Mittels der Kommandozeilenwerkzeuge von Kubernetes und Helm wird die Chart im Cluster ausgerollt. Daraufhin werden die entsprechenden Ressourcen erstellt, unter anderem der Pod für die Anwendung selbst und der Service zur Freigabe der Anwendung auf Port 8200. Das Kubernetes Kommandozeilenwerkzeug bietet die Möglichkeit lokal einen Port zu öffnen, an diesem ist dann ein Service im Cluster verfügbar. In diesem Fall wird zum Testen des Vault Servers der Port 8200 lokal geöffnet, um auf den Service zuzugreifen. War dieser Test erfolgreich, funktioniert die Helm Chart ohne Probleme und Vault wird innerhalb einer Enklave sicher ausgeführt und kann genutzt werden.

Um die Infrastruktur und die Ressourcen leichter verwalten zu können, wird Argo CD eingesetzt. Zunächst wird ein eigener Kubernetes *namespace* für Argo CD angelegt. Namespaces dienen der Separierung von Kubernetes Ressourcen. Mittels einer einzigen, durch Argo CD bereitgestellten, Konfigurationsdatei lässt sich Argo CD im Cluster installieren. Automatisch werden die benötigten Ressourcen erstellt. Die Abbildung 19 zeigt den Befehl zur Installation von Argo CD.

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

Abbildung 19: Argo CD: Installation (Quelle: eigene Darstellung)

Argo CD bietet die Möglichkeit Anwendungen zu erstellen, welche wiederum aus anderen Anwendungen bestehen können. Diese werden dann im Kubernetes Cluster nach den Vorgaben in den Konfigurationsdateien erstellt. Die komplette Anwendung dieser Arbeit wird aus folgenden Komponenten bestehen: Vault, Prometheus, Grafana und Traefik. Es wird ein GitHub Repository angelegt, in diesem werden die Konfigurationsdateien für die Struktur des Clusters abgelegt. Die Konfigurationsdatei *apps.yaml* der Gesamtanwendung zeigt auf einen Unterordner des GitHub Repositories, in diesem befinden sich die Dateien für die einzelnen Anwendungen und Ressourcen. Die *apps.yaml* ist im Anhang elf zu finden. Argo CD geht jede einzelne Datei in diesem Ordner durch. Zusätzlich wird die *syncPolicy selfHeal* aktiviert, sodass automatisch Ressourcen neu erstellt werden im Fehlerfall und der Zustand des Clusters stets mit den Vorgaben der Dateien in GitHub abgeglichen.

Zunächst wird eine Datei (Anhang 13) für die mittels EGo abgesicherte Vault Anwendung angelegt, diese zeigt auf das zuvor angelegte GitHub Repository und definiert, dass die Anwendung in einem eigenen *namespace*, *vault*, ausgerollt wird. Anschließend wird gesamte Konfigurationsdatei *apps.yaml* im Cluster deployed. Nach weniger Zeit sollte die Gesamtanwendung und die Vault Anwendung sichtbar in der Argo CD Oberfläche sein, daraufhin erfolgt ein weiterer Test des Vault Servers, ist dieser auch erfolgreich, werden die nächsten Komponenten angelegt. Die Abbildung 20 zeigt die Konfigurationsdatei.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: vault-ego
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: "https://github.com/WolfMa99/vault-ego-helm-chart.git"
    targetRevision: "main"
    path: templates
  destination:
    server: https://kubernetes.default.svc
    namespace: vault

  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Abbildung 20: EGo-Vault: Argo CD Applikation (Quelle: eigene Darstellung)

Für die Logging und Monitoring Komponente dieser Umsetzung werden Prometheus und Grafana genutzt. Prometheus stellt eine eigene Helm Chart zur Verfügung, welche Grafana bereits enthält und zur Ausführung in Kubernetes vorbereitet ist. Mittels der Konfigurationsdatei wird die Nutzung dieser Chart definiert. Dieser Logging und Monitoring Stack wird im *monitoring* Namespace ausgerollt. Die Konfigurationsdatei ist in Anhang 14 abgebildet.

Auch die Firma hinter Traefik stellt für ihren Reverse Proxy Helm Charts zur Verfügung. Solch eine wird auch in dieser Arbeit genutzt. In der Konfigurationsdatei zur Traefik Applikation wird definiert, dass diese im Namespace *traefik* installiert werden soll, zusätzlich, dass Zertifikate durch Let's Encrypt generiert werden sollen, alle HTTP Anfragen auf HTTPS umgeleitet werden und dass die Zertifikate persistent im Cluster gespeichert sind. Im Anhang 15 ist die zugehörige Konfigurationsdatei zu finden.

Damit sind alle Konfigurationsdateien für die Anwendungen selbst angelegt. Um die Funktionalität des Reverse Proxies zu nutzen, ist es notwendig diesen zu konfigurieren, damit die einzelnen Anwendungen abgesichert von außerhalb erreichbar sind. Traefik bietet verschiedene Möglichkeiten die Konfiguration vorzunehmen. Mit der Verwendung von Kubernetes ist der leichteste Weg das Anlegen der sogenannten *IngressRoute* Ressource. Sie erhält einen Namen sowie zusätzliche Optionen und wird im Namespace der entsprechenden Anwendung angelegt. Die dafür benötigten Konfigurationen liegen im gleichen Unterordner wie die der Anwendungen, sodass Argo CD die Ressourcen auch automatisch anlegt.

Innerhalb der Konfigurationen der IngressRoutes wird definiert, auf welchen Hostnamen sie reagieren und zu welchem Service mit welchem Port sie weiterleiten und welcher Dienst zur Erstellung der Zertifikate genutzt werden soll.

Die Grafana Oberfläche wird über die Subdomain *grafana.* erreichbar sein und verwendet den Service *prometheus-stack-grafana.* Die notwendige Konfiguration kann dem Anhang 16 entnommen werden.

Vault soll über die Subdomain *vault.* angesprochen werden können und zeigt auf dem *vault-ego-service* mit Port 8200. Die Konfiguration im Anhang 17 wird dafür genutzt.

Argo CD bietet eine HTTP Schnittstelle, aber auch ein Kommandozeileninterface, dafür wird ein gRPC Remote Procedure Calls (gRPC) Server genutzt, damit beide Schnittstellen über den gleichen Port, 443, funktionieren, ist eine Unterscheidung der Anfragen im Header notwendig. Diese Aufgabe übernimmt Traefik durch die dazugehörige Konfiguration. Anfragen an den gRPC Server werden durch einen bestimmten *Content-Type* im Header erkannt und mittels des Schemas *h2c* per HTTP2 weitergeleitet. Anfragen an den HTTP Server werden ganz normal behandelt. Argo CD ist erreichbar über die Subdomain

argocd.. Die Abbildung 21 zeigt die entsprechende Konfigurationsdatei, auch im Anhang 18.

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: argocd-server
  namespace: argocd
spec:
  entryPoints:
    - websecure
  routes:
    - kind: Rule
      match: Host(`argocd.bachelor.wcrw.de`)
      priority: 10
      services:
        - name: argocd-server
          port: 80
    - kind: Rule
      match: Host(`argocd.bachelor.wcrw.de`) && Headers(`Content-Type`, `application/grpc`)
      priority: 11
      services:
        - name: argocd-server
          port: 80
          scheme: h2c
  tls:
      certResolver: myresolver
```

Abbildung 21: Traefik: Argo CD Ingressroute (Quelle: eigene Darstellung)

Zusätzlich soll die Oberfläche des Reverse Proxies selbst von außen erreichbar sein, über die Subdomain *traefik.* und mittels Authentifizierung abgesichert werden. Dafür wird in der Konfiguration ein Kubernetes Secret angelegt, welches den Nutzernamen und das Passwort enthält, dazu eine sogenannte *Middleware*, diese zeigt, dass das Secret genutzt werden soll. Die eigentliche IngressRoute definiert die Nutzung der Middleware und welcher Service angesprochen werden soll. Um die Oberfläche aufzurufen, muss der Pfad */dashboard* angegeben werden, zur Nutzung der API der Pfad */api*. Um dieses Verhalten zu erreichen, wird die Konfiguration im Anhang 19 genutzt.

Der Reverse Proxy wurde zunächst so konfiguriert Let's Encrypt Zertifikate innerhalb der Staging Umgebung zu generieren, um die grundlegende Funktionalität zu testen. Damit die Dienste erreichbar sind und die Zertifikate generiert werden können, müssen die einzelnen Subdomains auf die öffentliche IP des Traefik Service zeigen, welche er durch den Azure Kubernetes Service zugeordnet bekommen hat. Wurden die Übergangszertifikate erfolgreich generiert und Zugriff auf die Oberflächen getestet, kann, durch Änderung der Konfiguration des Reverse Proxies, die Let's Encrypt Produktionsumgebung genutzt werden. Anschließend werden Zertifikate generiert, welche beispielsweise von Browsern ohne Warnung akzeptiert werden.

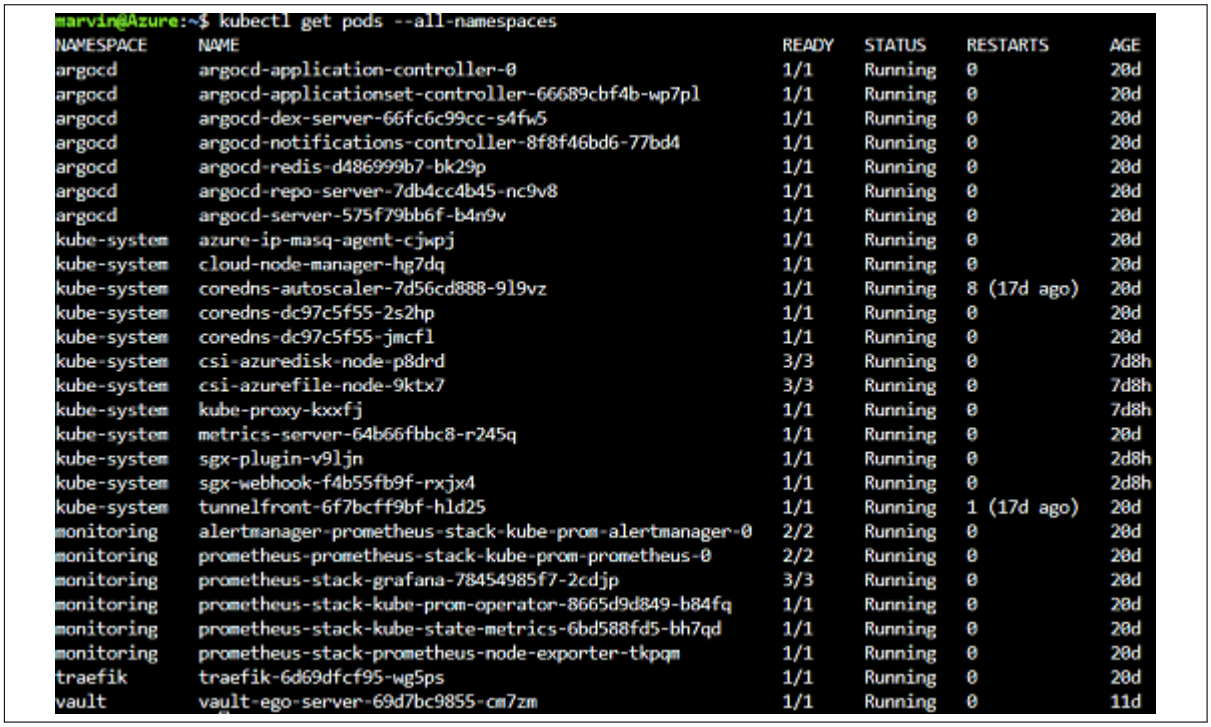
In der Grafana Oberfläche muss noch der Prometheus Server als Datenquelle ausgewählt werden, anschließend können die Metriken visualisiert werden. Die Abbildung 22 zeigt eine beispielhafte Visualisierung bestimmter Metriken im Bezug auf Kubernetes Cluster.

Abbildung 22: Azure Kubernetes Service Cluster: Grafana Dashboard (Quelle: eigene Darstellung)

7.3 Auswertung

In den vorhergehenden Abschnitten der Arbeit wurde zunächst das Konzept der Umsetzung beschrieben und danach die tatsächliche Implementation. Ziel war es Vault mit SGX abzusichern und in Kubernetes zur Ausführung zu bringen. Im Folgenden wird die Implementation ausgewertet.

Die Abbildung 23 zeigt alle laufenden Pods der Umsetzung.



```
marvin@Azure:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                                READY   STATUS    RESTARTS      AGE
argocd        argocd-application-controller-0                     1/1     Running   0             20d
argocd        argocd-applicationset-controller-66689cbf4b-wp7pl   1/1     Running   0             20d
argocd        argocd-dex-server-66fc6c99cc-s4fw5                  1/1     Running   0             20d
argocd        argocd-notifications-controller-8f8f46bd6-77bd4     1/1     Running   0             20d
argocd        argocd-redis-d486999b7-bk29p                        1/1     Running   0             20d
argocd        argocd-repo-server-7db4cc4b45-nc9v8                 1/1     Running   0             20d
argocd        argocd-server-575f79bb6f-b4n9v                      1/1     Running   0             20d
kube-system   azure-ip-masq-agent-cjwpj                           1/1     Running   0             20d
kube-system   cloud-node-manager-hg7dq                            1/1     Running   0             20d
kube-system   coredns-autoscaler-7d56cd888-9l9vz                  1/1     Running   8 (17d ago)   20d
kube-system   coredns-dc97c5f55-2s2hp                             1/1     Running   0             20d
kube-system   coredns-dc97c5f55-jmcfl                             1/1     Running   0             20d
kube-system   csi-azuredisk-node-p8drd                            3/3     Running   0             7d8h
kube-system   csi-azurefile-node-9ktx7                            3/3     Running   0             7d8h
kube-system   kube-proxy-kxxfj                                    1/1     Running   0             7d8h
kube-system   metrics-server-64b66fbbc8-r245q                     1/1     Running   0             20d
kube-system   sgx-plugin-v9ljn                                    1/1     Running   0             2d8h
kube-system   sgx-webhook-f4b55fb9f-rxjx4                         1/1     Running   0             2d8h
kube-system   tunnelfront-6f7bcff9bf-hld25                        1/1     Running   1 (17d ago)   20d
monitoring    alertmanager-prometheus-stack-kube-prom-alertmanager-0   2/2   Running   0          20d
monitoring    prometheus-prometheus-stack-kube-prom-prometheus-0  2/2     Running   0             20d
monitoring    prometheus-stack-grafana-78454985f7-2cdjp           3/3     Running   0             20d
monitoring    prometheus-stack-kube-prom-operator-8665d9d849-b84fq   1/1  Running   0             20d
monitoring    prometheus-stack-kube-state-metrics-6bd588fd5-bh7qd  1/1     Running   0             20d
monitoring    prometheus-stack-prometheus-node-exporter-tkpqm     1/1     Running   0             20d
traefik       traefik-6d69dfcf95-wg5ps                            1/1     Running   0             20d
vault         vault-ego-server-69d7bc9855-cm7zm                   1/1     Running   0             11d
```

Abbildung 23: Azure Kubernetes Service Cluster: laufende Pods (Quelle: eigene Darstellung)

Mittels EGo wurde der bestehende Programmcode kompiliert und zur Ausführung in SGX-Enklave vorbereitet. Die abgesicherte Anwendung wird in einem Docker Container ausgeführt und steht als Image öffentlich bereit. Innerhalb eines Kubernetes Clusters, auf der Azure Plattform, wird Vault ausgeführt. Zusätzlich werden Prometheus und Grafana genutzt, um Statistiken und Metriken zur Infrastruktur zu erheben. Die Oberflächen wurden erfolgreich mittels Traefik erreichbar gemacht, zusätzlich sind sie mit SSL-Zertifikaten von Let's Encrypt abgesichert.

Die Verwendung von Argo CD und einfach aufgebauten Konfigurationsdateien ermöglichen eine schnelle und simple Bereitstellung auf anderen Kubernetes Clustern, sodass die abgesicherte Vault Version vielseitig eingesetzt werden kann. Die Abbildung 24 gibt mittels Argo CD einen Überblick über die komplette Umsetzung inklusive aller Ressourcen.

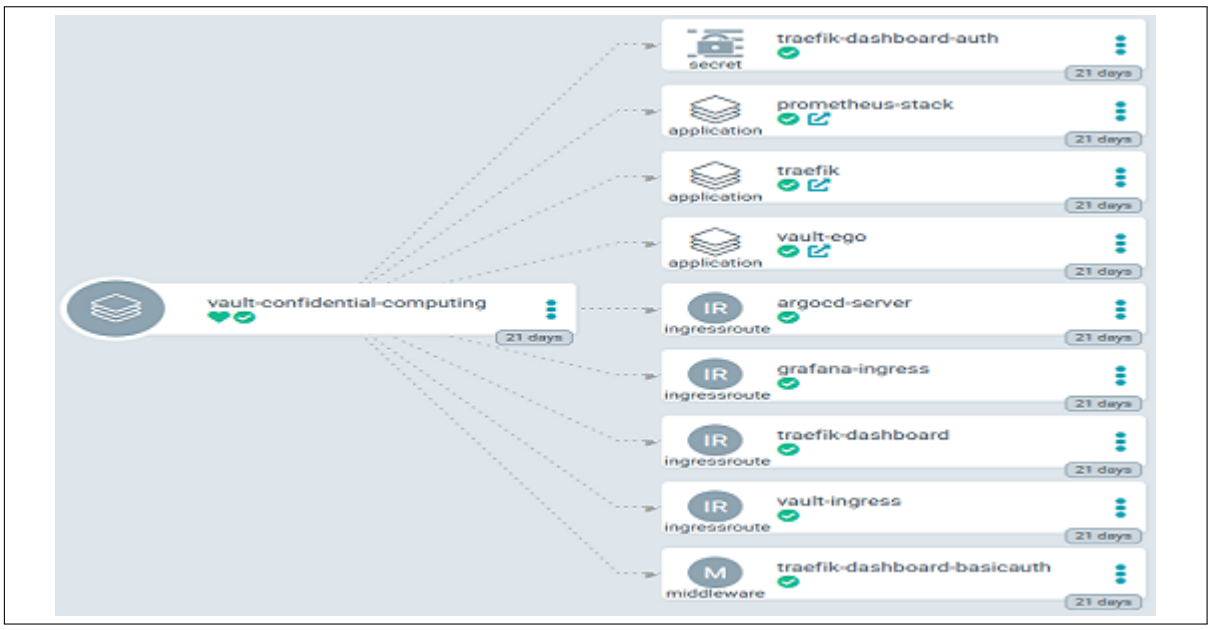


Abbildung 24: Argo CD: Überblick über Ressourcen (Quelle: eigene Darstellung)

Der Zugriff auf die, mit SGX, abgesicherte Variante von HashiCorp Vault geschieht genauso wie der Zugriff auf eine normale Installation von Vault. Die Anwendung verhält sich gleich, sodass für den Endanwender kein Unterschied erkennbar ist. Die Abbildung 25 zeigt, dass zunächst der entsprechende Host für den Vault Server gesetzt werden muss, anschließend der nötige Token, daraufhin kann der Status des Vault Servers ausgegeben werden.

```
marvinwolf@mvwl-sgx-vm:~$ export VAULT_ADDR=https://vault.bachelor.wcrw.de
marvinwolf@mvwl-sgx-vm:~$ export VAULT_TOKEN=mytoken
marvinwolf@mvwl-sgx-vm:~$ vault status
Key             Value
---             -----
Seal Type       shamir
Initialized     true
Sealed          false
Total Shares    1
Threshold       1
Version         1.12.0-dev1
Storage Type    inmem
Cluster Name    vault-cluster-de251baa
Cluster ID      802d13c6-2dc9-8b23-d784-31747e82d79d
HA Enabled      false
```

Abbildung 25: Vault-EGo: Status Test (Quelle: eigene Darstellung)

Die Verwendung von Open-Source-Technologien und Software bringen den Vorteil mit sich, dass der Programmcode und die Prozesse regelmäßig von einer Vielzahl an Entwicklern und Experten überprüft werden.

Die Implementation wurde erfolgreich schrittweise durchgeführt und das angestrebte Konzept umgesetzt.

8 Fazit und Ausblick

Das Ziel dieser Bachelorarbeit war es eine innovative Technologie wie Confidential Computing zu nutzen, um die bestehende Open-Source-Lösung HashiCorp Vault abzusichern, mit dem Ziel verschiedenste Sicherheitsprobleme zu lösen.

Theoretisch wurden verschiedene Cloud-Umgebungen sowie moderne Techniken, wie Container untersucht. Des Weiteren wurde die Geheimnisverwaltung in der Informationssicherheit betrachtet und ein verfügbares Werkzeug, um die Prozesse der Verwaltung und Überwachung weitestgehend zu vereinfachen, HashiCorp Vault. Der theoretische Schwerpunkt der Arbeit lag auf dem Gebiet des Confidential Computings, besonders auf der verwendeten Technologie Intel SGX. Zu dieser Technik wurde ein umfassender Einblick gegeben.

Praktisch wurde Intel SGX als ein Ansatz für Confidential Computing genutzt, um eine Anwendung innerhalb einer Enklave auszuführen. Die Enklave stellt einen abgesicherten Bereich des Speichers dar. Der Eigentümer der Maschine oder ein Angreifer innerhalb des Systems hat keinen Zugriff auf die verarbeiteten Daten in der Enklave, dadurch wird die Vertraulichkeit und Integrität gewahrt. Um weiteren Sicherheitsproblemen zu begegnen, kann HashiCorp Vault, eine Open-Source-Lösung, eingesetzt werden, um zum Beispiel Geheimnisse dynamisch zu verteilen und zu verwalten. Zu diesem Zweck wurde ein Konzept für die fertige Struktur erstellt und anschließend mithilfe von verschiedenen Open-Source-Werkzeugen implementiert und umgesetzt. Das Konzept wurde schrittweise aufgebaut und nach Abschluss jeden Schrittes getestet, sodass die Funktionalität sichergestellt ist.

Die Nutzung von Cloud-Infrastrukturen und Containern bringt einige Vorteile für den Anwender, wie kosteneffiziente Nutzung von Ressourcen und der Verringerung des Aufwandes der Verwaltung, deshalb wurde Docker als Container Plattform und Kubernetes zur Orchestrierung verwendet.

Die Arbeit zeigt, dass eine noch vergleichsweise junge Technologie, wie Intel SGX bereits gut erforscht und dokumentiert ist. Zudem ist die Migration von herkömmlichen Anwendungen leicht möglich. Der Aufwand der Migration ist abhängig vom Umfang der eigentlichen Anwendung, der Struktur oder des Programmcodes. Je nach verwendeter Programmiersprache existieren Arbeiten oder Werkzeuge, die einen Teil oder die gesamte Migration automatisieren, sodass die SGX-Funktionalität von Entwicklern schnell und einfach implementiert werden kann. Unter anderem bieten auch einige Unternehmen Dienstleistungen und Produkte an, um die Migration und Absicherung einer Anwendung zu übernehmen, dieser Weg richtet sich vorrangig an kommerzielle Kunden. Mittels des verwendeten Migrationsframeworks EGo können bestehende Open-Source-Projekte, welche in der Programmiersprache Go geschrieben sind, um SGX-Funktionen erweitert werden.

Die in den letzten Jahren immer größer werdende Technologie Blockchain ermöglicht die Zurechenbarkeit und Rückverfolgung von Handlungen. Jede Handlung kann dabei auf die Blockchain geschrieben werden, unzulässige Änderungen können erkannt werden und werden verworfen. Mittels kryptografischer Verfahren wird die Integrität gewahrt. Zukünftig können Haftungsfragen mittels dieser Technik schnell und zuverlässig beantwortet werden.[141] Die benötigte Software für eine Blockchain muss auf einer Vielzahl von Systemen installiert werden, die Verwaltung erfolgt mittels Choreografie, da sie ein gemeinsames Ziel verfolgen. Um Manipulationen an den Systemen und damit der Anwendung selbst zu verhindern, kann Confidential Computing eingesetzt werden, sodass der Betreiber keinen direkten Zugriff auf die Software und Daten selbst hat. Dabei kann die eigentliche Anwendung in den abgesicherten Enklaven ausgeführt werden. Ein ehemaliger Kollege unseres Teams, Dominik Lauck, verfasste 2019 seine Diplomarbeit zu dem Thema, wie die Technologie Confidential Computing im Zusammenspiel mit der Blockchain Technologie verwendet werden kann, um die Sicherheit und Privatsphäre in diesen Systemen zu verbessern. Zudem kann Confidential Computing die dezentrale Kommunikation absichern und die Vertraulichkeit von übertragenen Daten gewährleisten, was vordergründig in Sensornetzen wichtig ist oder bei der Nutzung von 5G in den einzelnen Zellen, aber auch zwischen den Zellen.[142]

Das Confidential Computing Consortium entwickelt und betreut verschiedene Werkzeuge und Umgebungen, um abgesicherte Anwendungen zu entwickeln. Dieser Entwicklungsprozess geschieht Open-Source, zudem sollen gewisse Standards und Richtlinien entstehen, um die Nutzung der Technologie weiter zu vereinfachen.

Insgesamt ist Confidential Computing ein sehr innovativer Ansatz, um Daten auf nicht vertrauenswürdigen Systemen zu verarbeiten, sodass der Anbieter der Systeme keinen Einblick in diese Daten erlangen kann und somit die Vertraulichkeit bewahrt wird. Die einfache Migration einer bestehenden Anwendung spricht für die Nutzung der Technologie Intel SGX und ebnet den Weg für Confidential Computing. Die Entwicklung und Bereitstellung von Anwendungen, insbesondere Micro-Services, auf dynamischen Infrastrukturen wird durch die Open-Source-Lösung HashiCorp Vault unterstützt, sodass Sicherheitsprobleme minimiert werden können und der Aufwand für Entwickler und Betreiber verringert wird.

[141] Vgl. Klymash, M./Beshley, M./Luntovskyy, A. (2022), S. 322 ff.
[142] Vgl. Luntovskyy, A. (2022).

Anhangverzeichnis

Anhang

Intel SGX SDK[143]

Switchless Calls[144]

Eleos[145]

Name	Intel SGX SDK	Switchless Calls	Eleos
TCB	?	?	1.000 Zeilen
Open Source	ja	ja	ja
Aufwand	sehr hoch	gering	sehr hoch
Syscall-Strategie	Standard	Workerpool mit Anpassung der Poolgröße, je nach Bedarf	Workerpool wird gesteuert durch Remote Procedure Calls
Syscall-Unterstützung	vollständig	vollständig	?
Besonderheiten	manuelle Umsetzung, Generierung von Proxy Functions mittels des Edger8r-Werkzeugs	seit 2018 im Intel SDK als zusätzlicher Standard vorhanden	EPC-Speicher kann durch eine zusätzliche Schicht verwaltet werden
Einschränkungen	keine	keine	nur für C++ umgesetzt, Einsatz eines veränderten SGX-Treibers erforderlich

Anhang 1: Migrationsansatz: Programmierbibliotheken (Quelle: eigene Darstellung)

[143] Intel Corporation (2020a).
[144] Tian, H. et al. (2018).
[145] Orenbach, M. et al. (2017).

Glamdring[146]

HotCalls[147]

EGo[148]

Name	Glamdring	HotCalls	EGo
TCB	6.000 Zeilen	150 Zeilen	?
Open Source	nein	nein	ja
Aufwand	hoch	gering	gering
Syscall-Strategie	Standard	ein Workerthread für die Ausführung der Syscalls, Kommunikation erfolgt über Hauptspeicher	Standard
Syscall-Unterstützung	komplett	komplett	komplett
Besonderheiten	Generierung des vertrauenswürdigen Codes durch das automatische Erkennen von markierten Variablen	Systemcalls werden automatisch erkannt und der dazugehörige Code wir generiert, dadurch ist eine schnellere Ausführung möglich	gesamte Anwendung wird innerhalb der Enklave ausgeführt
Einschränkungen	kann nur für C-Programme genutzt werden	keine	kann nur für Go-Programme genutzt werden

Anhang 2: Migrationsansatz: Automatisierungsframeworks (Quelle: eigene Darstellung)

146 Lind, J. et al. (2017).
147 Weisse, O./Bertacco, V./Austin, T. (2017).
148 Edgeless Systems GmbH (2022b).

SCONE[149]

Script Shield[150]

Name	SCONE	Script Shield
TCB	190.000 Zeilen	700 Zeilen + 80.000 Zeilen der *musl libc* Bibliothek
Open Source	nein	ja
Aufwand	gering	hoch
Syscall-Strategie	Workerthreads innerhalb des Kernelmoduls SCONE, asynchron	Standard
Syscall-Unterstützung	komplett *musl*-Bibliothek	eingeschränkt *musl*-Bibliothek, nur etwa 60 der über 300 Systemcalls werden unterstützt
Besonderheiten	Container-Anwendungen werden migriert, transparentes Verschlüsseln möglich, bereits migrierte Anwendungen zur Nutzung verfügbar	Migration des Interpretes von interpretierten Programm in die Enklave, danach Ausführung der Programmdateien
Einschränkungen	nur im Docker-Container-Format unterstützt	Quelltexte müssen sicher bereitgestellt werden können, bisher nur für JavaScript-, Lua- und Squirrel-Interpreter verfügbar

Anhang 3: Migrationsansatz: Shielding Layer (Quelle: eigene Darstellung)

[149] Arnautov, S. et al. (2016).
[150] Wang, H. et al. (2019).

Haven[151]

Graphene-SGX[152]

Occlum[153]

Name	Haven	Graphene-SGX	Occlum
TCB	5.600 Zeilen	1.300 Zeilen	15.000 Zeilen
Open Source	nein	ja	ja
Aufwand	sehr gering	sehr gering	gering
Syscall-Strategie	interne Abarbeitung, ansonsten Standard	interne Abarbeitung, ansonsten Standard	interne Abarbeitung, ansonsten Standard
Syscall-Unterstützung	komplett	komplett *glibc*-Bibliothek	eingeschränkt *musl*-Bibliothek
Besonderheiten	speziell für Windows-Anwendungen	Nachladen von Bibliotheken dynamisch möglich, Integritätschecks, Container-Migration ist möglich	Möglichkeit für sicheres Multitasking innerhalb der Enklave durch Isolation des Prozesses, Verifikation der Anwendung bevor sie startet, migrierte Anwendungen können innerhalb von Containern ausgeführt werden
Einschränkungen	Bisher nur Prototyp	Nur Docker-Container-Format wird unterstützt, veränderte SGX-Treiber notwendig oder Patches am Kernel	Nur Docker-Container-Format wird unterstützt, spezielles Kernelmodul wird benötigt

Anhang 4: Migrationsansatz: Library OS (Quelle: eigene Darstellung)

[151] Baumann, A./Peinado, M./Hunt, G. (2014).
[152] Tsai, C.-c./Porter, D. E./Vij, M. (2017).
[153] Shen, Y. et al. (2020).

Listing 1: Vault-EGo: Dockerfile

```
FROM ubuntu:18.04 AS base

WORKDIR /

RUN apt update -y && apt upgrade -y
RUN apt install -y wget git build-essential software-properties-common \
libssl-dev

RUN wget -qO- https://download.01.org/intel-sgx/sgx_repo/
ubuntu/intel-sgx-deb.key | apt-key add
RUN add-apt-repository "deb [arch=amd64] https://download.01.org/
intel-sgx/sgx_repo/ubuntu `lsb_release -cs` main"

RUN wget https://github.com/edgelesssys/ego/releases/download/
v0.5.0/ego_0.5.0_amd64.deb
RUN apt install -y ./ego_0.5.0_amd64.deb

RUN git clone https://github.com/hashicorp/vault

EXPOSE 8200

ADD script.sh /
RUN chmod +x /script.sh
CMD ./script.sh
```

Listing 2: Vault-EGo: script.sh

```
#!/bin/bash

cd vault
ego-go build -v -o bin/vault
cd bin
ego sign vault
ego run vault server -dev -dev-no-store-token -dev-root-token-id mytoken
-dev-listen-address="0.0.0.0:8200"
```

Listing 3: Vault-EGo: Helm Chart.yaml

```
apiVersion: v2
name: vault-ego-helm-chart
description: Vault EGo Helm Chart
version: 0.1.0
type: application
appVersion: 0.1.0
home: https://github.com/WolfMa99/vault-ego-helm-chart.git
keywords:
- cloudnative
- vault
- ego
- helm
- kubernetes
```

Listing 4: Vault-EGo: Helm deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: vault-ego-server
namespace: vault
labels:
app: vault-ego-app
spec:
replicas: 1
selector:
matchLabels:
app: vault-ego-app
template:
metadata:
labels:
app: vault-ego-app
spec:
tolerations:
- key: kubernetes.azure.com/sgx_epc_mem_in_MiB
operator: Exists
effect: NoSchedule
containers:
- name: vault-ego-helm-chart
image: marvinwolf/ego-vault-test:latest
imagePullPolicy: IfNotPresent
ports:
- containerPort: 8200
resources:
limits:
kubernetes.azure.com/sgx_epc_mem_in_MiB: 100
```

Listing 5: Vault-EGo: Helm service.yaml

```
apiVersion: v1
kind: Service
metadata:
name: vault-ego-service
namespace: vault
spec:
selector:
app: vault-ego-app
ports:
- protocol: TCP
port: 8200
targetPort: 8200
type: ClusterIP
```

Listing 6: Vault-Confidential Computing App: Ordnerstruktur

```
└── vault-confidential-computing
    │   apps.yaml
    │
    └── helm
        │   Chart.yaml
        │
        └── templates
                argocd-ingress.yaml
                grafana-ingress.yaml
                monitoring.yaml
                traefik-dashboard-ingress-basicauth.yaml
                traefik.yaml
                vault-ego.yaml
                vault-ingress.yaml
```

Listing 7: Vault-Confidential Computing App: apps.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
name: vault-confidential-computing
namespace: argocd
finalizers:
- resources-finalizer.argocd.argoproj.io
spec:
project: default
source:
repoURL: https://github.com/WolfMa99/vault-confidential-computing
targetRevision: HEAD
path: helm
destination:
server: https://kubernetes.default.svc
namespace: production

syncPolicy:
automated:
prune: true
selfHeal: true
```

Listing 8: Vault-Confidential Computing App: Chart.yaml

```
apiVersion: v1
description: Application environmnet
name: prodcution
version: "0.1.0"
```

Listing 9: Vault-Confidential Computing App: vault-ego.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
name: vault-ego
namespace: argocd
finalizers:
- resources-finalizer.argocd.argoproj.io
spec:
project: default
source:
repoURL: "https://github.com/WolfMa99/vault-ego-helm-chart.git"
targetRevision: "main"
path: templates
destination:
server: https://kubernetes.default.svc
namespace: vault

syncPolicy:
automated:
prune: true
selfHeal: true
```

Listing 10: Vault-Confidential Computing App: monitoring.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
name: prometheus-stack
namespace: argocd
finalizers:
- resources-finalizer.argocd.argoproj.io
spec:
project: default
source:
repoURL: "https://prometheus-community.github.io/helm-charts"
chart: kube-prometheus-stack
targetRevision: 35.4.2
helm:
values: |
prometheusOperator:
tls:
enabled: false
admissionWebhooks:
enabled: false
destination:
server: https://kubernetes.default.svc
namespace: monitoring

syncPolicy:
automated:
prune: true
selfHeal: true
```

Listing 11: Vault-Confidential Computing App: traefik.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
name: traefik
namespace: argocd
finalizers:
- resources-finalizer.argocd.argoproj.io
spec:
project: default
source:
repoURL: "https://helm.traefik.io/traefik"
chart: traefik
targetRevision: 10.20.0
helm:
values: |
additionalArguments:
- --certificatesresolvers.myresolver.acme.tlschallenge
- --certificatesresolvers.myresolver.acme.email=wolfma99@gmail.com
- --certificatesresolvers.myresolver.acme.storage=/certs/acmeprod.json
- --certificatesresolvers.myresolver.acme.caserver=https://acme-v02.api.
letsencrypt.org/directory
ports:
web:
redirectTo: websecure
ingressRoute:
dashboard:
enabled: false
persistence:
enabled: true
path: /certs
size: 128Mi

destination:
server: https://kubernetes.default.svc
namespace: traefik

syncPolicy:
automated:
prune: true
```

```
selfHeal: true
```

Listing 12: Vault-Confidential Computing App: grafana-ingress.yaml

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
name: grafana-ingress
namespace: monitoring

spec:
entryPoints:
- websecure
routes:
- match: Host(`grafana.bachelor.wcrw.de`)
kind: Rule
services:
- name: prometheus-stack-grafana
port: 80
tls:
certResolver: myresolver
```

Listing 13: Vault-Confidential Computing App: vault-ingress.yaml

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
name: vault-ingress
namespace: vault

spec:
entryPoints:
- websecure
routes:
- match: Host(`vault.bachelor.wcrw.de`)
kind: Rule
services:
- name: vault-ego-service
port: 8200
tls:
certResolver: myresolver
```

Listing 14: Vault-Confidential Computing App: argocd-ingress.yaml

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
name: argocd-server
namespace: argocd
spec:
entryPoints:
- websecure
routes:
- kind: Rule
match: Host(`argocd.bachelor.wcrw.de`)
priority: 10
services:
- name: argocd-server
port: 80
- kind: Rule
match: Host(`argocd.bachelor.wcrw.de`) &&
Headers(`Content-Type`, `application/grpc`)
priority: 11
services:
- name: argocd-server
port: 80
scheme: h2c
tls:
certResolver: myresolver
```

Listing 15: Vault-Confidential Computing App: traefik-dashboard-ingress-basicauth.yaml

```
---
apiVersion: v1
kind: Secret
metadata:
name: traefik-dashboard-auth
namespace: traefik

data:
users: a2FuZ29yb286JGFwcjEkdGlQbFBINXYkYlJrUHBSUlYuYUxUWnhFRzdYbmduMAoK

---
apiVersion: traefik.containo.us/v1alpha1
kind: Middleware
metadata:
name: traefik-dashboard-basicauth
namespace: traefik

spec:
basicAuth:
secret: traefik-dashboard-auth

---
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
name: traefik-dashboard
namespace: traefik

spec:
entryPoints:
- websecure
routes:
- match: Host(`traefik.bachelor.wcrw.de`) && (PathPrefix(`/dashboard`)
|| PathPrefix(`/api`))
kind: Rule
middlewares:
- name: traefik-dashboard-basicauth
namespace: traefik
services:
```

```
- name: api@internal
kind: TraefikService
tls:
certResolver: myresolver
```

Literatur

Adamski, A. (2018a): *Overview of Intel SGX. Part 1, SGX Internals*, in: https://blog.quarkslab.com/overview-of-intel-sgx-part-1-sgx-internals.html, (02. 06. 2022).

Adamski, A. (2018b): *Overview of Intel SGX. Part 2 SGX Externals*, in: https://blog.quarkslab.com/overview-of-intel-sgx-part-2-sgx-externals.html, (02. 06. 2022).

Advanced Micro Devices, Inc. (2020): *AMD SEV-SNP. Strengthening VM Isolation with Integrity Protection and More*, Santa Clara.

Advanced Micro Devices, Inc. (2021): *AMD64 Architecture Programmer's Manual Volume 2: System Programming. Revision 3.38*, Santa Clara.

Amazon Web Services (2022): *About AWS*, in: https://aws.amazon.com/about-aws/, (02. 06. 2022).

Anati, I. et al. (2013): *Innovative Technology for CPU Based Attestation and Sealing*, Santa Clara.

Argo Project Authors (2022): *Argo CD. Declarative continuous delivery with a fully-loaded UI.* in: https://argoproj.github.io/cd/, (02. 06. 2022).

ARM Limited (2009): *ARM Security Technology. Building a Secure System using TrustZone® Technology*, Cherry Hinton.

Arnautov, S. et al. (2016): *SCONE: Secure Linux Containers with Intel SGX*, in: Association, U. (Hrsg.), *12th USENIX Symposium on Operating Systems Design and Implementation*, Savannah.

Asylo Authors (2021): *Asylo*, in: https://asylo.dev/, (02. 06. 2022).

Baumann, A./Peinado, M./Hunt, G. (2014): *Shielding Applications from an Untrusted Cloud with Haven*, in: Association, U. (Hrsg.), *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Broomfield.

Beningo, J. (2020): *Wie man TrustZone zur Sicherung von IoT-Geräten mit minimaler Hardware-Komplexität und Kosten verwendet*, in: https://www.digikey.de/de/articles/how-to-use-trustzone-to-secure-iot-devices/, (07. 06. 2022).

Brickell, E./Li, J. (2007): *Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities*, in: Computing Machinery, A. (Hrsg.), *WPES '07: Proceedings of the 2007 ACM workshop on Privacy in electronic society*, New York.

Bulck, J. V. et al. (2018): *Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution*, in: Association, U. (Hrsg.), *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore.

Bundesamt für Soziale Sicherung (2021): *APP.bd.6 Secrets Management mit Hashicorp Vault*, Bonn.

Confidential Computing Foundation (2022): *What is the Confidential Computing Consortium?*, in: https://confidentialcomputing.io/, (02. 06. 2022).

Costan, V./Devadas, S. (2016): *Intel SGX Explained*, in: *IACR Cryptol. ePrint Arch* 1, S. 1–118.

Costan, V./Lebedev, I./Devadas, S. (2017): *Secure Processors Part I: Background, Taxonomy for Secure Enclaves and Intel SGX Architecture*, in: Jongh, M. (Hrsg.), *Foundations and Trends in Electronic Design Automation: Vol. 11: No. 1-2*, Boston.

Daemen, J./Rijmen, V. (2001): *The Design of Rijndael. AES - The Advanced Encryption Standard*, Nijmegen.

Diffie, W./Hellman, M. E. (1976): *New directions in cryptography*, in: The Institute of Electrical und Electronics Engineers, I. (Hrsg.), *IEEE Transactions on Information Theory 22.6*, Cambridge, S. 644–654.

Docker Inc. (2021): *Swarm mode overview*, in: `https://docs.docker.com/engine/swarm/`, (07. 06. 2022).

Edgeless Systems GmbH (2022a): *Edgeless Systems. Turn the public cloud into your private cloud.* in: `https://www.edgeless.systems/`, (07. 06. 2022).

Edgeless Systems GmbH (2022b): *EGo. linux-sgx-driver*, in: `https://github.com/edgelesssys/ego/`, (07. 06. 2022).

Fortanix (2022): *Fortanix. Security, whereever your data is...* in: `https://www.fortanix.com/`, (07. 06. 2022).

Free Software Foundation, Inc. (2022): *The GNU C Library (glibc)*, in: `https://www.gnu.org/software/libc/`, (02. 06. 2022).

Gentry, C. (2009): *A Fully Homomorphic Encryption Scheme*, Diss. Stanford University. Stanford.

Georgia Institute of Technology (2019a): *SGX Bootstrap. Overview*, in: `https://sgx101.gitbook.io/sgx101/sgx-bootstrap/`, (07. 06. 2022).

Georgia Institute of Technology (2019b): *SGX Bootstrap. Enclave*, in: `https://sgx101.gitbook.io/sgx101/sgx-bootstrap/enclave/`, (07. 06. 2022).

Georgia Institute of Technology (2019c): *SGX Bootstrap. Attestation*, in: `https://sgx101.gitbook.io/sgx101/sgx-bootstrap/attestation/`, (07. 06. 2022).

Grafana Labs (2022): *Grafana. The open observability platform*, in: `https://grafana.com/`, (02. 06. 2022).

HashiCorp, Inc (2022a): *Architecture*, in: `https://www.vaultproject.io/docs/internals/architecture/`, (02. 06. 2022).

HashiCorp, Inc (2022b): *Encryption as a Service: Transit Secrets Engine*, in: `https://learn.hashicorp.com/tutorials/vault/eaas-transit/`, (02. 06. 2022).

HashiCorp, Inc (2022c): *HCL*, in: `https://github.com/hashicorp/hcl/`, (02. 06. 2022).

HashiCorp, Inc (2022d): *PKI Secrets Engine*, in: `https://www.vaultproject.io/docs/secrets/pki/`, (02. 06. 2022).

HashiCorp, Inc (2022e): *Seal/Unseal*, in: `https://www.vaultproject.io/docs/concepts/seal/`, (02. 06. 2022).

HashiCorp, Inc (2022f): *What is Vault?*, in: `https://www.vaultproject.io/docs/what-is-vault/`, (02. 06. 2022).

Heidkamp, P. et al. (2020): *Cloud Monitor 2020. Die Integrationsfähigkeit und Interoperabilität der Cloud stärken*, Köln.

Helm Authors (2022): *Helm. What is Helm?*, in: `https://helm.sh/`, (02. 06. 2022).

Hunt, R. (2001): *PKI and digital certification infrastructure*, in: The Institute of Electrical und Electronics Engineers, I. (Hrsg.), *Proceedings. Ninth IEEE International Conference on Networks, ICON 2001*, Bangkok.

Intel Corporation (2020a): *Intel Software Guard Extensions (Intel SGX) SDK for Linux OS. Developer Reference. Version 2.9.1*, Santa Clara.

Intel Corporation (2020b): *Intel® Processors Voltage Settings Modification Advisory*, in: `https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00289.html`, (02. 06. 2022).

Intel Corporation (2020c): *Intel® Software Guard Extensions (Intel® SGX) SDK for Windows* OS. Developer Reference. Version 2.7.* Santa Clara.

Intel Corporation (2020d): *Intel® Trust Domain Extensions. White Paper*, Santa Clara.

Intel Corporation (2021a): *Intel(R) Software Guard Extensions for Linux* OS. linux-sgx-driver*, in: `https://github.com/intel/linux-sgx-driver/`, (07. 06. 2022).

Intel Corporation (2021b): *Kann eine Intel® Software Guard Extensions (Intel® SGX) innerhalb einer Intel® Trust-Gastdomäne (Intel® TD) ausgeführt werden?*, in: `https://www.intel.de/content/www/de/de/support/articles/000088208/software/intel-security-products.html`, (07. 06. 2022).

Intel Corporation (2021c): *Q3 2018 Speculative Execution Side Channel Update*, in: `https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html`, (02. 06. 2022).

Intel Corporation (2022a): *Build an Intel® Software Guard Extensions ECDSA Attestation Service to Strengthen Enclave Security. Intel® Provisioning Certification Service for ECDSA Attestation*, in: `https://api.portal.trustedservices.intel.com/provisioning-certification/`, (02. 06. 2022).

Intel Corporation (2022b): *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4. Version 077*, Santa Clara.

Intel Corporation (2022c): *Intel® Trust Domain Extensions (Intel® TDX)*, in: `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html`, (07. 06. 2022).

Internet Engineering Task Force (IETF) (2018): *The Transport Layer Security (TLS) Protocol Version 1.3*, Fremont.

Johnson, S. P. (2018): *An update on 3rd Party Attestation*, in: `https://www.intel.com/content/www/us/en/developer/articles/technical/an-update-on-3rd-party-attestation.html`, (07. 06. 2022).

Kaplan, D. (2017): *PROTECTING VM REGISTER STATE WITH SEV-ES*, Santa Clara.

Kaplan, D./Powell, J./Woller, T. (2021): *AMD MEMORY ENCRYPTION. Version 9*, Santa Clara.

Klymash, M./Beshley, M./Luntovskyy, A. (2022): *Future Intent-Based Networking. On the QoS Robust and Energy Efficient Heterogeneous Software Defined Networks*, Cham.

Kochovski, A. (2022): *The Risks and Benefits of Cloud Storage in 2022. ...* in: https://www.cloudwards.net/the-risks-and-benefits-of-cloud-storage/, (02. 06. 2022).

Let's Encrypt (2022a): *About Let's Encrypt*, in: https://letsencrypt.org/about/, (02. 06. 2022).

Let's Encrypt (2022b): *Staging Environment*, in: https://letsencrypt.org/docs/staging-environment/, (02. 06. 2022).

Lind, J. et al. (2017): *Glamdring: Automatic Application Partitioning for Intel SGX*, in: Association, U. (Hrsg.), *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara.

Luntovskyy, A. (2021a): *Teil II. Rechnernetze. Vorlesungsskript*, BA Dresden.

Luntovskyy, A. (2021b): *Verteilte Systeme. Vorlesungsskript*, BA Dresden.

Luntovskyy, A. (2022): *Challenges of 5G and Beyond Mobile Radio Networks. 10th IEEE MRW-2022 Conft (accepted for publication)*, Gdansk.

Luntovskyy, A./Gütter, D. (2020): *Moderne Rechnernetze. Protokolle, Standards und Apps in kombinierten drahtgebundenen, mobilen und drahtlosen Netzwerken*, Dresden.

Luntovskyy, A./Gütter, D. (2022): *Highly-Distributed Systems. IoT, Robotics, Mobile Apps, Energy Efficiency, Security*, Dresden.

McClain, K. (2022): *musl libc. a new libc striving to be fast, simple, lightweight, free and correct*, in: https://wiki.musl-libc.org/, (02. 06. 2022).

McKeen, F. et al. (2013): *Innovative Instructions and Software Model for Isolated Execution*, in: Computing Machinery, A. (Hrsg.), *HASP '13: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, New York.

McKeen, F. et al. (2016): *Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave*, in: Computing Machinery, A. (Hrsg.), *HASP 2016: Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, New York.

Mechalas, J. P. (2016): *Intel® Software Guard Extensions Part 7: Refine the Enclave with Proxy Functions. Overview*, in: https://www.intel.com/content/www/us/en/developer/articles/training/intel-software-guard-extensions-tutorial-part-7-refining-the-enclave.html, (07. 06. 2022).

Mechalas, J. P. (2017): *Properly Detecting Intel Software Guard Extensions (Intel SGX) in Your Applications*, in: https://www.intel.com/content/www/us/en/developer/articles/technical/properly-detecting-intel-software-guard-extensions-in-your-applications.html, (07. 06. 2022).

Mechalas, J. P./Odom, B. J. (2016): *Intel Software Guard Extensions Tutorial Series: Part 1, Intel SGX Foundation.* in: https://www.intel.com/content/www/us/en/developer/articles/training/intel-software-guard-extensions-tutorial-part-1-foundation.html, (02. 06. 2022).

Mechalas, J. P./Reed, I. (2016): *Intel Software Guard Extensions Tutorial Series: Part 3, Design an Application*, in: https://www.intel.com/content/www/us/en/developer/

articles/training/software-guard-extensions-tutorial-series-part-3.html, (02. 06. 2022).

Mell, P./Grance, T. (2011): *The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology*, Gaithersburg.

Microsoft (2022): *Was ist Azure?*, in: https://azure.microsoft.com/de-de/overview/what-is-azure/, (02. 06. 2022).

Minkin, M. (2018): *Improving Performance and Security of Intel SGX*, Haifa.

Murdock, K. et al. (2020): *Plundervolt: Software-based Fault Injection Attacks against Intel SGX*, in: The Institute of Electrical und Electronics Engineers, I. (Hrsg.), *2020 IEEE Symposium on Security and Privacy (SP)*, San Francisco.

Naehrig, M./Lauter, K./Vaikuntanathan, V. (2011): *Can Homomorphic Encryption be Practical?*, in: Computing Machinery, A. (Hrsg.), *CCSW '11: Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, New York.

Nam, D. (2019): *API Design Implications of Boilerplate Client Code*, in: The Institute of Electrical und Electronics Engineers, I. (Hrsg.), *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego.

Newman, S. (2019): *Monolith To Microservices. Evolutionary Patterns to Transform Your Monolith*, Sebastopol,

Orenbach, M. et al. (2017): *Eleos: ExitLess OS Services for SGX Enclaves*, in: Computing Machinery, A. (Hrsg.), *EuroSys '17: Proceedings of the Twelfth European Conference on Computer Systems*, New York.

Paul McDonald (2008): *Introducing Google App Engine + our new blog*, in: http://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html, (02. 06. 2022).

Priebe, C. et al. (2019): *SGX-LKL: Securing the Host OS Interface for Trusted Execution*, in: *ArXiv* 1, S. 1–17.

Prometheus Authors (2022): *Prometheus. Monitoring system & time series database*, in: https://prometheus.io/, (02. 06. 2022).

Rao, A. (2022): *Rising to the Challenge - Data Security with Intel Confidential Computing*, in: https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Rising-to-the-Challenge-Data-Security-with-Intel-Confidential/post/1353141, (02. 06. 2022).

Rivest, R. L./Shamir, A./Adleman, L. M. (1978): *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, in: Computing Machinery, A. (Hrsg.), *Communications of the ACM. Volume 21. Issue 2*, New York.

Russinovich, M. (2018): *Azure Confidential Computing*, in: https://azure.microsoft.com/de-de/blog/azure-confidential-computing/, (02. 06. 2022).

Sabt, M./Achemlal, M./Bouabdallah, A. (2015): *Trusted Execution Environment: What It is, and What It is Not*, in: The Institute of Electrical und Electronics Engineers, I. (Hrsg.), *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki.

Scarlata, V. et al. (2019): *Supporting Third Party Attestation for Intel® SGX with Intel® Data Center Attestation Primitives*, Santa Clara.

Schuster, F. (2021): *Confidential-Computing soll eine neue, sichere Cloud-Ära einläuten. Datenverarbeitung in sicheren Enklaven*, in: `https://www.security-insider.de/confidential-computing-soll-eine-neue-sichere-cloud-aera-einlaeuten-a-1010505/`, (02. 06. 2022).

Shamir, A. (1979): *How to Share a Secret*, in: Computing Machinery, A. (Hrsg.), *Communications of the ACM. Volume 22. Issue 11*, New York.

Shen, Y. et al. (2020): *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, in: Computing Machinery, A. (Hrsg.), *ASPLOS 2020: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York.

The Inclavare Containers Authors (2021): *Inclavare Containers*, in: `https://inclavare-containers.io/en/`, (02. 06. 2022).

The kernel development community (2022): *31. Software Guard eXtensions (SGX)*, in: `https://www.kernel.org/doc/html/latest/x86/sgx.html`, (07. 06. 2022).

The Kubernetes Authors (2022): *What is Kubernetes? Why you need Kubernetes and what it can do*, in: `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`, (07. 06. 2022).

Tian, H. et al. (2018): *Switchless Calls Made Practical in Intel SGX*, in: Computing Machinery, A. (Hrsg.), *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, New York.

Traefik Labs (2022): *Traefik. The Cloud Native Application Proxy*, in: `https://traefik.io/traefik/`, (02. 06. 2022).

Tsai, C.-c./Porter, D. E./Vij, M. (2017): *Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX*, in: Association, U. (Hrsg.), *2xx7 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara.

Vaucher, S. et al. (2018): *SGX-Aware Container Orchestration for Heterogeneous Clusters*, in: The Institute of Electrical und Electronics Engineers, I. (Hrsg.), *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Wien.

Wang, H. et al. (2019): *Running Language Interpreters Inside SGX: A Lightweight,Legacy-Compatible Script Code Hardening Approach*, in: Computing Machinery, A. (Hrsg.), *Asia CCS '19: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Auckland.

Weisse, O./Bertacco, V./Austin, T. (2017): *Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves*, in: Computing Machinery, A. (Hrsg.), *ISCA '17: Proceedings of the 44th Annual International Symposium on Computer Architecture*, New York.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Zustimmung des Partnerunternehmens in der Praxis zur Verwendung betrieblicher Unterlagen habe ich eingeholt. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Dresden, 15. Juli 2022 Unterschrift des Verfassers